micro procesoare

Cristian LUPU

2 · 4 · 8

Seria
MICROPROCESOARE

Cristian Lupu

MICROPROCESOARE 2/4/8 BIȚI

EDITURA MILITARĂ - BUCUREŞTI, 1995

ISBN 973-32-0409-9

Cuprins

CUVÂNT ÎNAINTE

Seria MICROPROCESOARE, care debutează cu această carte, își propune să prezinte cititorilor problematica primului nivel de utilizare a microprocesoarelor, nivel ce se constituie din descrierea circuitelor, inclusiv a celor auxiliare, a familiilor de microprocesoare, amănunte legate de asamblarea acestor componente, programarea lor și legăturile cu nivelurile superioare hardware și software. Abordarea subiectului va fi făcută atât din punct de vedere hardware, cât și software, în cele mai multe situații vizându-se aspecte concrete, practice. Aspectul general al lucrărilor din această serie va fi însă cu precădere HARDWARE. Ilustrăm, într-un fel, o mai veche părere a proiectanților de calculatoare, cu care de fapt nu mai suntem întru totul de acord, sau, mai bine zis, pe care o interpretăm altfel astăzi: "întâi e hardware-ul, apoi software-ul". În fond, nu încercăm decât să materializăm ideea necesității unei literaturi tehnice care să acopere nivelul de realizare efectivă a unei arhitecturi electronice (de calcul sau de control).

Seria MICROPROCESOARE se adresează mai multor categorii de cititori. O primă categorie este cea a proiectanților de structuri cu microprocesoare destinate încorporării lor în diverse aplicații. Pentru acești cititori lucrările, ghidate așa cum am mai spus pe o linie practică, își propun ca pe măsura aparițiilor să-i orienteze pe proiectanți, implicit sau explicit, în mai multe direcții: ce microprocesor să aleagă pentru a realiza un proiect optim, ce circuite auxiliare îi mai trebuie, ce acoperire software este necesară.

O altă grupă de cititori cărora dorim să ne adresăm este cea a utilizatorilor unor echipamente complexe având la bază microprocesoare. Aici pot intra cei care beneficiază sau se ocupă de întreținerea PC-urilor, echipamentelor de automatizare, a altor sisteme de calcul sau control realizate cu microprocesoare. Deși pentru această categorie de cititori microprocesoarele nu mai sunt direct accesibile, ele fiind în general "acoperite" de mai multe straturi software, considerăm că lucrările pe care le vom prezenta pot fi utile în special prin descrierile arhitecturale, așa cum, de exemplu, cărțile despre motoare pot fi

utile conducătorilor de autovehicule. Mai ales în cazurile de funcționare greșită sau de defectare. Pe de altă parte, descrierile arhitecturale — seturi de instrucțiuni, moduri de adresare, registre generale, întreruperi, dar și datele referitoare la performanțele electronice — viteze, tehnici de asamblare, oferă acestei categorii posibilități de apreciere comparativă a "puterii" mașinilor pe care le utilizează și/sau a impactului pe care microprocesorul îl are asupra caracteristicilor mașinii. Din acest punct de vedere microprocesorul reprezintă unul din criteriile principale de apreciere a echipamentului care îl încorporează.

În sfârșit, o a treia categorie de cititori cărora ne adresăm este a celor interesați în evoluția microprocesoarelor. Încercând să acoperim o gamă cât mai largă, de la microprocesoare bit-slice *pe 2/4 biți la microprocesoare* single-chip *pe 8/16/32/64 biți, sau de la microcalculatoare* single-chip *la* transputer-*e și alte structuri RISC, sperăm să realizăm o imagine clară a dezvoltării acestei laturi a tehnicii de calcul sau, mai general, de procesare a informației, prin prezentarea multitudinii aspectelor legate de perfecționările tehnologice, arhitecturale și de programare care caracterizează apariția fiecărei clase sau generații de microprocesoare.*

În sensul celor spuse mai sus credem că era firesc să începem seria anunțată reluând prezentarea celor mai cunoscute și utilizate microprocesoare pe 2/4/8 biți și anume microprocesoarele bit-slice *Intel 3000 și Am 2900 și microprocesorul* single-chip *Z80. Deși aceste microprocesoare nu mai reprezintă dispozitive de mare performanță, ele sunt încă utilizate, constituind în plus, după părerea noastră, o bună introducere în problematică, având și rolul de a familiariza cititorii cu structura și funcțiile unui microprocesor.*

Lucrarea imediat următoare va fi o prezentare profesională a unei răspândite utilizări a microprocesorului Z80, microcalculatorul Sinclair ZX Spectrum. Seria va continua cu titluri ce vor acoperi evoluții tehnologice — de exemplu cea care a permis creșterea mărimii cuvântului la 16/32/64 de biți — sau se vor referi la evoluții arhitecturale, cum sunt cele datorate filozofiei RISC, o a treia lucrare pregătită deja pentru apariție ocupându-se de primul microprocesor pe 16 biți larg folosit, Intel 8086.

Apreciind încă o dată Editura Militară pentru modul colegial, prietenesc chiar, dar și, cum se spune astăzi, de adevărat parteneriat, cu care abordează întreaga activitate de elaborare a cărților, ne manifestăm speranța că seria pe care o lansăm acum va fi utilă celor ce doresc să-și însușească cunoștințe despre utilizarea și evoluția atât de dinamică și interesantă a microprocesoarelor.

Cristian Lupu

1

MICROPROGRAMARE ŞI MICROPROCESOARE *BIT-SLICE*

1.1. APARIŢIA ŞI DEZVOLTAREA METODEI MICROPROGRAMĂRII

1.1.1. ISTORIC

Conceptul de microprogramare, ca metodă de proiectare a structurilor de control numerice, este prezentat pentru prima dată în iulie 1951 de profesorul Maurice Wilkes de la Universitatea din Cambridge, la o conferinţă de calculatoare ţinută la Universitatea din Manchester. În comunicarea sa [1], Wilkes enunţa ideile principale care stau la baza microprogramării şi propunea primul model (fig. 1.1) al unei structuri de control microprogramate:

"... să considerăm controlul propriu-zis, adică partea maşinii care generează impulsurile necesare validării porţilor asociate registrelor aritmetice

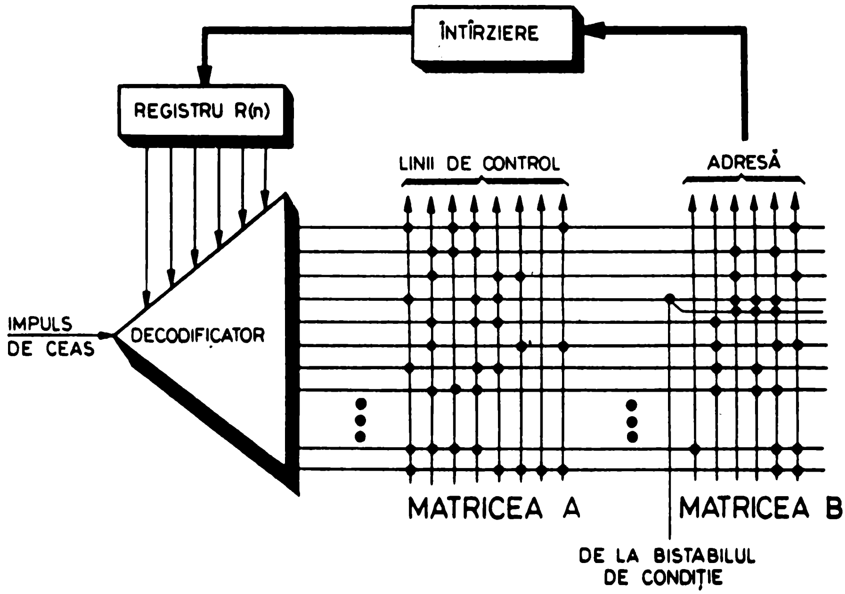


Fig. 1.1. Modelul structurii de control microprogramate propus de Maurice Wilkes

sau de control. Proiectantul acestei părţi din maşină acţionează de obicei într-o manieră ad-hoc, proiectând scheme-bloc până când ajunge la un aranjament care îi satisface cerinţele tehnice şi pare a fi acceptabil din punct de vedere economic. Aş dori să sugerez o cale prin care controlul poate fi proiectat mai *sistematic* (s.n.) şi de aceea mai puţin complicat.

Orice operaţie apelată de un ordin care corespunde codului de ordin (instrucţiunii, n.n.) al maşinii presupune o *secvenţă de paşi* (s.n.) ce poate cuprinde transferuri între memorie şi registrele de *control* sau aritmetice şi transferuri între registre. Fiecare din aceşti paşi este realizat prin pulsarea unor anumite semnale asociate cu registrele de control sau aritmetice.

Voi numi aceşti paşi *microoperaţii*. O operaţie-maşină propriu-zisă (instrucţiune, n.n.) este deci compusă dintr-o secvenţă sau un *microprogram* de microoperaţii.

Figura 4 (aici fig. 1.1, n.n.) reprezintă o schemă cu ajutorul căreia se obţin microoperaţiile. Impulsul care generează o microoperaţie-intră în arborele de decodificare şi este condus la una din ieşiri în funcţie de conţinutul registrului R. El trece prin matricea A şi generează impulsuri la unele din ieşirile matricei în funcţie de aranjamentul impedanţelor de cuplaj. Aceste impulsuri validează porţile asociate registrelor de control sau aritmetice şi deci realizează o microoperaţie. Impulsul generat în arborele de decodificare trece de asemenea şi prin matricea B generând impulsuri la unele din ieşirile acestei matrice. Aceste impulsuri sunt conduse printr-o scurtă linie de întârziere la registrul R schimbând conţinutul acestuia. Ca rezultat, următorul impuls care intră în arbore va fi condus la alte ieşiri şi în consecinţă va realiza o altă microoperaţie. Se vede deci că fiecare rând din matricea A corespunde unei microoperaţii din secvenţa necesară (din microprogram, n.n.) realizării unei operaţii-maşină".

După cum se vede, Wilkes a observat că instrucţiunile-maşină pot fi realizate ca secvenţe de mai multe operaţii elementare care să specifice transferul de informaţie între diferitele elemente ale unei unităţi centrale. Cu ajutorul unor porţi de control proiectantul maşinii poate comanda circulaţia informaţiei între aceste elemente. Aceasta înseamnă că fiecare poartă trebuie să fie validată printr-o linie de comandă a cărei stare, 0 sau 1, poate fi memorată într-o matrice de tip *read-only* nedistructivă. În primul model al lui Wilkes a fost folosită o matrice cu ferite cablată permanent ca un set de porţi SAU care activa un subset de linii de control reprezentând ieşirea propriu-zisă a matricei. Schema originală propusă de Wilkes (fig 1.1) consta din matricea de control A, matricea de secvenţiere B, un arbore de decodificare şi logica de decizie.

Decodificatorul acceptă ieşirea registrului R ca intrare, decodificând *n* biţi (mărimea registrului de adresă R). În acest fel va fi validată o singură linie din cele 2^n linii de ieşire ale matricei de decodificare.

Matricea de control A constă dintr-un set de linii orizontale (cele 2^n ieşiri ale decodificatorului) şi un alt set de linii verticale (liniile de control). Punctele negre reprezintă impedanţe de cuplare între cele două seturi. Rezultă că orice semnal electric care se propagă de-a lungul unei linii orizontale va fi conectat

la linia verticală, dacă există o impedanţă de cuplare. Porţile de control din sistem sunt astfel validate prin conectarea fiecăreia la una dintre liniile verticale din matricea A.

O linie orizontală poate fi concepută acum ca o *microinstrucţiune*. Când este selectată, ea va activa un set predeterminat de linii de control.

Impulsul care iese din decodificator trece, de asemenea, şi prin matricea de secvenţiere B. Liniile de ieşire ale acestei matrice poziţionează un set de elemente de memorare care formează registrul de adresă R. Linia de întârziere are rolul de a asigura prelungirea perioadei în care informaţia de control este stabilă, după schimbarea conţinutului registrului de adresă R.

O microinstrucţiune va fi deci selectată corespunzător adresei din registrul de adresă R. Microinstrucţiunea va genera impulsuri de validare corespunzând codului de control stocat în acel cuvânt şi va selecta adresa microinstrucţiunii ce va controla starea maşinii în ciclul următor.

Logica de decizie şi salt, necesară oricărei structuri de control, este asigurată de un bistabil de condiţie, a cărui ieşire depinde de realizarea unui anumit test din sistem. În funcţie de rezultatul testului (reuşit sau nereuşit) bistabilul de condiţie selectează una din cele două căi alternative în microprogram. În figura 1.2 se dă schema logică utilizată pentru execuţia deciziei de secvenţiere şi selectarea adresei microinstrucţiunii următoare.

Modelul propus de Wilkes folosea deci două matrice: una pentru specificarea microoperaţiilor executate într-un ciclu, cealaltă pentru determinarea adresei microinstrucţiunii următoare. Cu ajutorul conceptelor de *microoperaţie*, *microinstrucţiune* şi *microprogram* Wilkes definea în esenţă metoda: microprogramarea înseamnă controlul unei structuri numerice prin intermediul unor cuvinte "citite" secvenţial, pas cu pas, dintr-o memorie. Prin citirea succesivă a acestor cuvinte, microinstrucţiunile, se generează semnalele de control, microoperaţiile, necesare funcţionării corecte a structurii respective. Proiectarea unei structuri microprogramate este astfel mult mai sistematică şi mai flexibilă decât cea a unei structuri convenţionale realizate prin logică cablată.

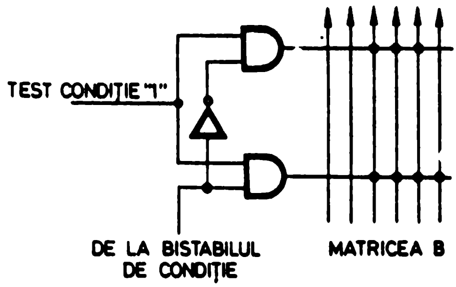


Fig. 1.2. Schema logică utilizată de Maurice Wilkes pentru execuţia deciziei de secvenţiere şi selectarea adresei următoare

După primele investigaţii făcute de Wilkes, microprogramarea a primit o oarecare atenţie în deceniul 1950-1960 dar, de fapt, ea a fost ţinută pe planul al doilea, nefiind încă utilizată comercial. De abia după lansarea în 1965 de către IBM a seriei de calculatoare System/360, microprogramarea, care servise ca metodă de implementare a majorităţii modelelor acestei serii, va ieşi din anonimat, răspândindu-se din ce în ce mai mult. Evoluţia corespunde de fapt

dezvoltării tehnologice, afirmarea cu întârziere a microprogramării datorându-se în principal prețului prohibitiv al memoriei de control (matricele A și B din modelul Wilkes). Până la mijlocul anilor 1960 avantajele simplității și flexibilității oferite de microprogramare au fost cu mult contracarate de dezavantajul mare al timpului de acces la memoria de microinstrucțiuni.

Primele dezvoltări ale tehnologiilor de fabricare a memoriei de control care au influențat microprogramarea au fost cele ale memoriilor liniare rezistive, capacitive sau inductive. Au urmat dispozitivele neliniare cu diode, ferite, cu film magnetic. De asemenea s-au realizat dispozitive de memorare optice, cu fibre optice, memorii holografice [4].

Microprogramarea a primit un nou impuls odată cu apariția memoriilor semiconductoare rapide și ieftine și îndeosebi după fabricarea primelor memorii fixe integrate (Fairchild Corporation anunța prima memorie fixă, realizată în tehnologie MOS, în 1967). La începutul deceniului 1970-1980 microprogramarea a fost utilizată și în domeniul minicalculatoarelor, răspândindu-se astfel foarte mult, pe o bază cu adevărat comercială.

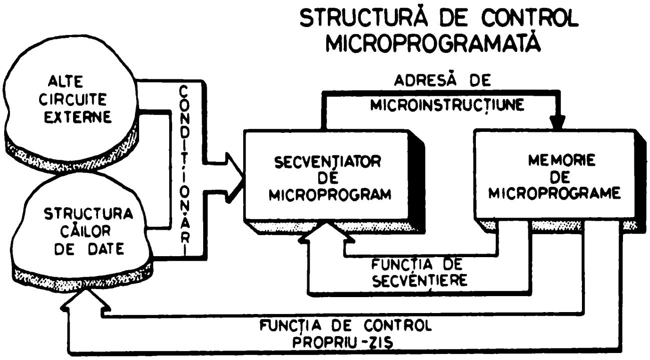
O nouă direcție în dezvoltarea microprogramării este marcată de realizarea circuitelor integrate pe scară largă. Este vorba de apariția memoriilor fixe LSI și mai ales a microprocesoarelor de tip *bit slice* microprogramabile (în 1973 firma National Semiconductor lansează un circuit de 4 biți, General Purpose Controller/Processor, care putea fi folosit ca element constructiv pentru lungimi de cuvânt de până la 32 de biți, iar firma Rockwell fabrică tot atunci un procesor paralel pe 4 biți, de asemenea microprogramabil, cu un set de 50 de instrucțiuni și un ciclu de 5μs). Aceste progrese tehnologice au avut o influență deosebită asupra microprogramării, care a devenit o metodă uzuală de proiectare a structurilor de control numerice, ieșind din sfera exclusivă a proiectanților de calculatoare și găsindu-și aplicații în cele mai diverse domenii.

1.1.2. DEFINIREA UNEI STRUCTURI DE CONTROL MICROPROGRAMATE

Deși microprogramarea a fost propusă ca o metodă de implementare a instrucțiunilor de limbaj-mașină, deci ca o metodă de proiectare a unității de control dintr-un calculator, conceptul de microprogramare a evoluat căpătând un sens mai general. Pentru a ilustra această evoluție, vom da ca exemplu o structură de control microprogramată realizată cu un secvențiator de adresă de tip Am 2909 (fig. 1.3 și 1.4).

În principiu, așa cum s-a văzut și în § 1.1, o *mașină microprogramată* este o mașină în care o secvență coerentă de microinstrucțiuni, deci un microprogram, este folosită pentru execuția operațiilor mari, macrooperațiilor, care definesc funcționarea mașinii. Dacă mașina este o unitate centrală a unui calculator, fiecare secvență de microinstrucțiuni va fi deci folosită pentru executarea unei (macro)instrucțiuni. Zona de memorare a acestor secvențe de

Fig. 1.3.
O maşină microprogramată



microinstrucţiuni sau a microprogramelor este de obicei numită *memorie de microprograme* sau *memorie de control.*

O maşină numerică se poate împărţi în două părţi care grupează căile de circulaţie a datelor, respectiv căile de control. Una din tehnicile de implementare a structurii căilor de control este şi microprogramarea. În figura 1.3 se prezintă componenţa în mare a structurii microprogramate a căilor de control şi relaţiile globale cu restul maşinii numerice microprogramate. După cum se vede, structura de control microprogramată este alcătuită din două elemente principale: secvenţiatorul de microprograme care generează adresa de microinstrucţiune şi memoria de microprograme.

Structura de control microprogramată va avea două funcţii principale: funcţia de control propriu-zis şi funcţia de secvenţiere. Conform acestora, microinstrucţiunea, elementul de control efectiv, are două părţi care definesc de fapt *controlul prin microprogram.* Acestea se referă la:

– definirea şi controlul tuturor *microoperaţiilor* care trebuie realizate în maşină;

– definirea şi controlul *adresei microinstrucţiunii următoare.*

Definirea diferitelor microoperaţii care trebuie executate în maşină cuprinde de obicei selecţia operanzilor unităţii aritmetice-logice (UAL), funcţia UAL, destinaţia UAL, controlul transportului, controlul deplasării, controlul întreruperilor, controlul I/E, în general, controlul tuturor resurselor hardware ale maşinii.

Definirea adresei microinstrucţiunii următoare se referă la identificarea sursei pentru adresa următoare, la controlul condiţiilor de test, uneori la generarea directă a valorii adresei.

Structura de control microprogramată poate fi, ca aceea din figura 1.4, alcătuită dintr-o memorie de microprograme, un registru de microinstrucţiune, un secvenţiator de adresă de microprogram, o memorie de tip PROM pentru controlul acestui secvenţiator, un multiplexor de condiţii şi o altă memorie PROM de *mapare* pentru generarea adreselor de început ale secvenţelor corespunzătoare macrooperaţiilor pe care trebuie să le execute maşina. Memoria de microprograme conţine secvenţe de microinstrucţiuni, compuse la rândul lor

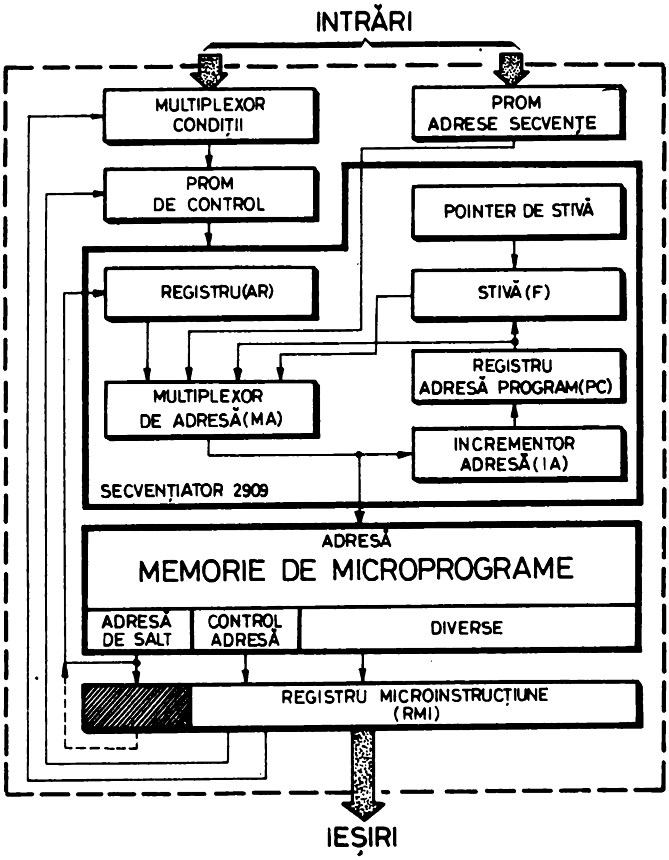


Fig. 1.4. O structură de control microprogramată realizată cu un secvențiator 2909

din microoperații reprezentând operațiile primitive ce trebuie să le realizeze hardware-ul mașinii. Memoria de microprograme corespunde în acest fel matricelor A și B din modelul propus de Wilkes.

Structura din figura 1.4 funcționează în felul următor:

– microinstrucțiunea adresată de secvențiatorul 2909 este citită din memoria de microprograme în registrul de microinstrucțiune RMI;

– microoperațiile care intră în alcătuirea microinstrucțiunii aflate în RMI sunt decodificate și utilizate ca informație de control;

– informația de control obținută astfel stimulează resursele hardware corespunzătoare, efectuând operațiile primitive din mașină, microoperațiile;

– secvențiatorul 2909 utilizează informația de stare rezultată prin decodificare în PROM-ul de control, împreună cu una din intrările în multiplexorul de adresă, pentru a genera adresa microinstrucțiunii următoare.

Prin repetarea acestui proces se va executa secvența de microinstrucțiuni, deci microprogramul, care controlează funcționarea mașinii.

Structura de control microprogramată prezentată în figura 1.4 este o structură sincronă de tip *pipeline*. O asemenea structură suprapune execuția

microinstrucţiunii curente cu extragerea din memoria de microprograme a microinstrucţiunii următoare. Elementul cheie al unei astfel de structuri este registrul *pipeline* (notat RMI în figura 1.4) plasat la ieşirea memoriei de microprograme. Această tehnică, utilizată pentru îmbunătăţirea performanţelor de timp ale unei structuri de control microprogramate, va fi detaliată în § 1.1.3.

Secvenţiatorul 2909, aşa cum se vede în figura 1.4, este alcătuit dintr-un multiplexor de adresă MA cu 4 intrări, un registru AR, stiva pentru bucle şi subrutine F, registrul-adresă-program PC şi incrementorul de adresă IA. Se observă că adresa următoare poate fi selectată cu ajutorul multiplexorului MA care are următoarele intrări: intrarea directă D, registrul AR, registrul program PC şi stiva F. Structura şi funcţionarea secvenţiatorului 2909 vor fi descrise amănunţit în § 1.2.3. Totuşi, pentru înţelegerea complexităţii funcţiei de secvenţiere a unei structuri microprogramate vom menţiona că, dacă pentru controlul adresei următoare se foloseşte un câmp de 3 biţi, cu ajutorul acestuia pot fi implementate următoarele microoperaţii:

CONTINUARE	adresa următoare este adresa precedentă incrementată cu 1;
SALT	adresa următoare este adresa din registrul AR;
SALT PE CONDIŢIE	adresa următoare este adresa din registrul AR, atunci când condiţia selectată este adevărată; în caz contrar, secvenţa continuă;
SALVARE PC	adresa din PC este introdusă în stivă;
SALT LA SUBRUTINĂ	adresa următoare este adresa din registrul AR, adresa curentă este memorată în stivă;
ADRESĂ DE START	adresa următoare este adresa generată de PROM-ul pentru adrese de secvenţe (fig. 1.4);
REVENIRE	adresa următoare este adresa memorată ultima oară în stivă;
TEST SFÂRŞIT BUCLĂ	adresa următoare este adresa memorată ultima oară în stivă când condiţia selectată este adevărată; în caz contrar, secvenţa continuă.

Folosind aceste microoperaţii de secvenţiere, se pot scrie uşor microprograme destul de complicate necesare controlului unor diverse maşini numerice.

Structura microprogramată descrisă mai sus poate fi privită şi ca o *cutie neagră* ale cărei intrări sunt macrooperaţiile şi condiţiile de test, iar ieşirile - câmpurile de control din microinstrucţiune. Funcţiile realizate, cea de control propriu-zisă şi cea de secvenţiere, sunt definite prin microprogramul stocat în memoria de microprograme. O astfel de structură, pe care o vom numi mai departe *structură de control microprogramată*, *SCM*, poate fi utilizată pentru implementarea funcţiei de control din orice maşină numerică, bineînţeles cu particularităţile, avantajele şi dezavantajele ce vor fi discutate în capitolele următoare.

Evoluând astfel spre generalitate, o *structură de control microprogramată* poate fi definită ca un automat finit destinat implementării funcţiei de control din sisteme care prelucrează informaţia, folosind tehnica microprogramării. O asemenea structură, al cărei prim model a fost propus de Maurice Wilkes, se caracterizează în general prin:

– organizarea memoriei de control:
– formatul microinstrucţiunii;
– implementarea microinstrucţiunii.

În continuare se vor prezenta pe larg aceste caracteristici ale SCM.

1.1.3. CARACTERISTICI ALE STRUCTURILOR DE CONTROL MICROPROGRAMATE

1.1.3.1. **Organizarea memoriei de microprograme**

S-a definit mai sus că memoria de microprograme sau memoria de control este o unitate de memorie de mare viteză în care se găsesc microprogramele în execuţie. Această memorie este de două feluri: de tip ROS, R*ead* O*nly* S*torage*, sau ROM, R*ead* O*nly* M*emory*, şi de tip WCS, W*ritable* C*ontrol* S*tore*. Memoria ROS este fixă pentru o structură dată şi nu poate fi modificată sub controlul microprogramului. Schimbarea conţinutului memoriei ROS este deci echivalentă, sub aspect funcţional, cu schimbarea structurii. În cazul memoriei WCS, conţinutul ei poate fi modificat sub controlul microprogramului. Astfel, cu ajutorul unei memorii WCS, o SCM poate fi modificată funcţional într-un mod dinamic.

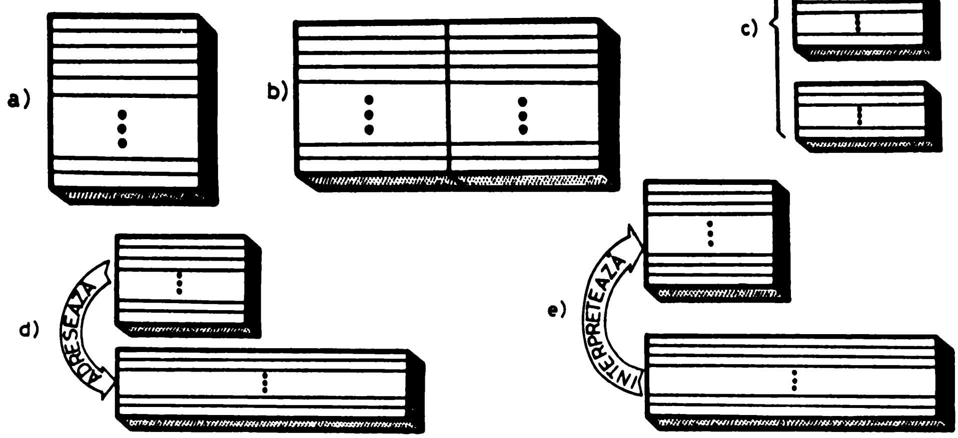
Memoria de microprograme a unei SCM, indiferent de caracterul ei ROS şi/sau WCS, poate fi organizată logic în mai multe moduri (figura 1.5).

1. Cea mai simplă şi mai obişnuită structură de memorie de microprograme este cea în care fiecărui cuvânt de memorie îi corespunde o singură microinstrucţiune (fig. 1.5*a*). Citirea unei microinstrucţiuni presupune un singur acces la memoria de control.

2. O altă structură de memorie de microprograme este aceea în care fiecare cuvânt de memorie conţine două sau mai multe microinstrucţiuni (figura 1.5*b*). Avantajul acestei scheme este că în registrul de microinstrucţiune se citesc simultan două sau mai multe microinstrucţiuni, micşorându-se în acest fel numărul de referinţe la memoria de microprograme, ceea ce conduce în final la mărirea vitezei de lucru a SCM.

3. Într-o altă structură, memoria de microprograme este împărţită în blocuri (figura 1.5*c*). Pentru acest tip de memorie există două feluri de adrese de microinstrucţiune: adrese de microinstrucţiune din acelaşi bloc cu microinstrucţiunea curentă şi adresele altor blocuri. Ca un rezultat al acestei organizări adresele microinstrucţiunilor din acelaşi bloc sunt mai scurte decât cele corespunzătoare unei structuri fără blocuri. Împărţirea memoriei de microprograme pe blocuri se poate face ţinând cont atât de structura microprogramului,

Fig. 1.5. Organizări ale memoriei de microprograme: cu o microinstrucţiune pe cuvânt (a), cu două microinstrucţiuni pe cuvânt (b), structurată pe blocuri (c), divizată (d), structurată pe două niveluri – nanoprogramare (e)



cât şi de circuitele de memorie disponibile. O organizare de acest fel conduce în general la micşorarea microinstrucţiunii.

4. Memoria de microprograme divizată (figura 1.5*d*) este alcătuită din două unităţi de memorie distincte, cu dimensiuni de cuvânt diferite. Unitatea de memorie cu dimensiune de cuvânt mai mică va conţine microinstrucţiuni mai simple (de exemplu, transferuri de informaţie între registre, transferuri de informaţie între microinstrucţiune şi registre sau/şi iniţierea execuţiei unei microinstrucţiuni rezidente în cealaltă unitate de memorie). A doua unitate de memorie conţine microinstrucţiuni mai lungi, care pot controla simultan mai multe resurse hardware ale maşinii. Această structură de memorie de microprograme divizată poate să conducă la micşorarea volumului de memorie faţă de o organizare standard, atunci când pot fi executate frecvent microinstrucţiuni scurte sau/şi mai multe microinstrucţiuni scurte se referă la aceeaşi microinstrucţiune lungă.

5. Memoria de microprograme poate fi structurată pe două niveluri (figura 1.5*e*), astfel încât microinstrucţiunile din unitatea de memorie de nivel mai scăzut, numite *nanoinstrucţiuni*, să interpreteze microinstrucţiunile din unitatea de memorie de nivel superior la fel cum, de exemplu, microinstrucţiunile interpretează instrucţiunile de limbaj-maşină într-o unitate centrală microprogramată. Această tehnică a *nanoprogramării* este deci echivalentă conceptual cu microprogramarea, structura pe două niveluri oferind o flexibilitate în proiectarea microinstrucţiunilor de la nivelul superior analogă cu proiectarea instrucţiunilor de limbaj-maşină.

Toate aceste organizări ale memoriei de microprograme au fost utilizate în minicalculatoare [2] cu scopul de a micşora volumul ocupat de microprogramele de control sau/şi de a mări viteza lor de execuţie. Trebuie spus, totuşi, că, în general, structurile de control microprogramate construite cu microprocesoare

bit-slice folosesc organizarea cea mai simplă: o microinstrucţiune pe cuvânt (figura 1.5*a*). Memoria de microprograme va fi deci o memorie de N cuvinte a câte M biţi cu locaţii definite în general continuu de la 0 la N-1.

1.1.3.2. **Formatul microinstrucţiunii**

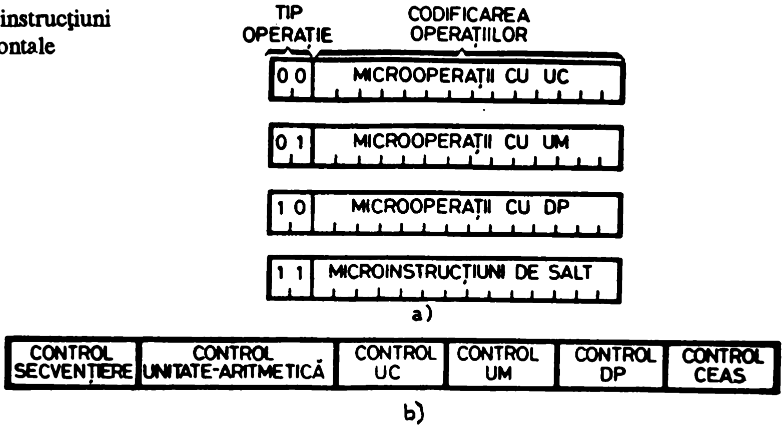
Împărţirea cuvântului de microprogram în diferite zone de control numite *câmpuri* defineşte *formatul* microinstrucţiunii.

Pentru caracterizarea formatului microinstrucţiunii se pot utiliza mai mulţi parametri: caracteristica de verticalitate-orizontalitate, modul de control, gradul de codificare.

Deosebit de important în proiectarea microinstrucţiunii este numărul de resurse hardware ale maşinii a cărei funcţie de control se doreşte să fie implementată printr-o SCM. În acest sens microinstrucţiunile se clasifică de obicei ca *verticale* sau *orizontale*, deşi aceşti termeni se referă la extremităţile unei game largi de structuri. Microinstrucţiunile verticale efectuează operaţii simple, singulare, de tipul încărcare, adunare, memorare, salt etc. Acest tip de microinstrucţiune se aseamănă în principiu cu instrucţiunile de limbaj maşină care conţin un cod-operaţie şi unul sau mai mulţi operanzi (microprogramarea verticală se mai numeşte şi *microprogramare soft* [3]). Lungimea microinstrucţiunilor verticale variază, de obicei, între 12-24 biţi. Microinstrucţiunile orizontale controlează foarte multe resurse hardware care funcţionează în paralel. O singură microinstrucţiune orizontală ar putea să controleze funcţionarea simultană şi independentă a uneia sau a mai multor UAL, accesul la memoria principală, generarea condiţionată a adresei următoare, diverse registre de lucru, bistabili de stare etc. Menţionăm că, deşi microprogramarea orizontală, numită uneori *microprogramare hard* [3], are avantajul potenţial al utilizării eficiente a hardware-ului, scrierea şi punerea la punct a microprogramelor orizontale, optimale din punctul de vedere al utilizării resurselor (inclusiv memoria de microprograme), este o problemă destul de dificilă. Microinstrucţiunile orizontale au o lungime tipică de 64 biţi. Aşadar, vom considera caracteristica de verticalitate-orizontalitate a unei microinstrucţiuni ca fiind impusă de numărul de resurse hardware din maşină controlate *simultan* de microinstrucţiunea respectivă.

Arhitectura unei maşini nu determină în mod necesar şi complet proiectarea microinstrucţiunii ca verticală sau orizontală: o aceeaşi arhitectură poate fi controlată cu un microprogram lung, lent, scris cu microinstrucţiuni verticale sau cu un microprogram scurt, rapid, scris cu microinstrucţiuni orizontale. În figura 1.6*a* şi *b* se dau două formate de microinstrucţiune, vertical respectiv orizontal, pentru controlul unui canal selector microprogramat [15]. Repertoriul de microinstrucţiuni verticale cuprinde patru tipuri de operaţii codificate într-un câmp de doi biţi. Aceste tipuri de operaţii mari corespund controlului prin microprogram al uneia din principalele resurse hardware ale maşinii, în acest caz blocurile de logică cu UC, UM, DP şi structura de control

Fig. 1.6. Microinstrucţiuni verticale şi orizontale



SCM. Microoperaţiile, specifice tipului de operaţie, pot fi codificate corespunzător în restul de biţi ai microinstrucţiunii. Microinstrucţiunea orizontală propusă este împărţită în câmpuri separate pentru controlul *simultan* al principalelor resurse hardware ale maşinii.

Dezvoltarea continuă în domeniul tehnologiei circuitelor a oferit proiectantului de structuri de control microprogramate blocuri constructive din ce în ce mai complexe. Această situaţie a făcut ca puterea unei microoperaţii să fie într-o permanentă creştere, astfel încât ceea ce la început putea fi considerat ca format din mai multe microoperaţii distincte să poată fi ulterior înglobat într-o singură microoperaţie. Din acest motiv linia de despărţire între microinstrucţiuni verticale şi orizontale nu a putut fi bine definită. Nu lipsit complet de umor, termenul microinstrucţiunii *diagonale* salvează această situaţie. Astfel de microinstrucţiuni combină cele mai bune caracteristici ale microinstrucţiunilor verticale şi orizontale: sunt uşor de înţeles, generat şi implementat, ca şi microinstrucţiunile verticale,. având în acelaşi timp o capacitate crescută de control simultan al resurselor hardware ale maşinii.

Microinstrucţiunile descrise până acum au ilustrat *controlul imediat*: microoperaţiile sunt convertite în semnale de control care acţionează direct şi imediat resursele maşinii. O altă tehnică de microprogramare, numită *control rezidual*, foloseşte unele registre preîncărcate pentru controlul resurselor hardware. Într-o schemă cu control rezidual microoperaţiile nu mai controlează direct resursele. Acest rol va fi îndeplinit de registrele de control încărcate anterior. Valoarea unui asemenea registru preîncărcat, controlul rezidual, poate reprezenta microoperaţia pe care trebuie să o realizeze o anumită unitate funcţională, de exemplu adresa unui registru sau a unui cuvânt de memorie. Microinstrucţiunile pot să înlocuiască sau să modifice valoarea unuia sau mai multor registre de control, aşa ca în figura 1.7. Această tehnică a controlului rezidual asigură o economie de memorie de microprograme, atunci când unele unităţi funcţionale realizează repetat aceeaşi operaţie. Registrele de control pot fi manevrate cu ajutorul unor microinstrucţiuni de tip vertical, asigurându-se în

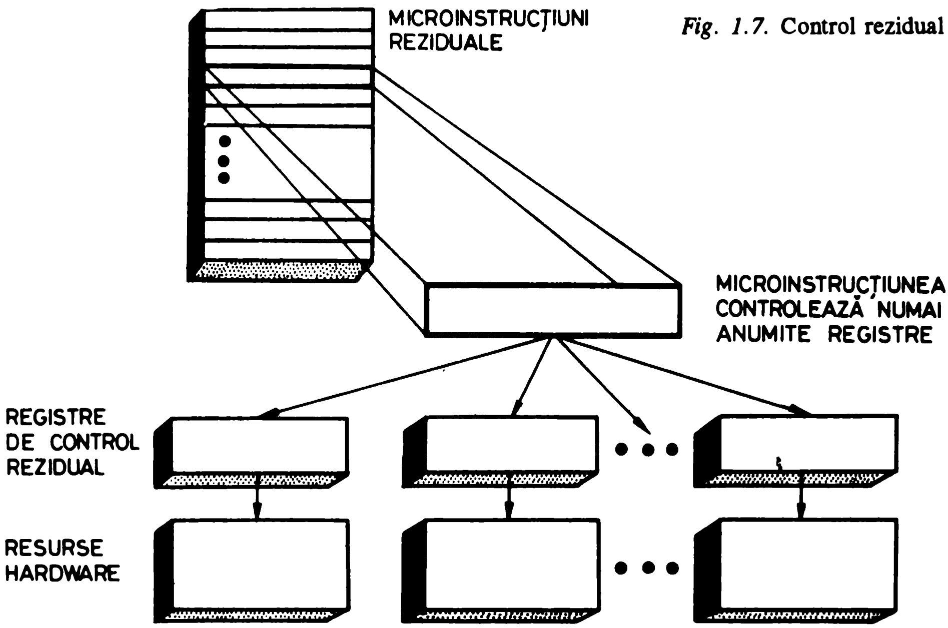


Fig. 1.7. Control rezidual

acelaşi timp controlul simultan al mai multor resurse hardware, ca şi prin microinstrucţiunile orizontale.

Lungimea unei microinstrucţiuni depinde şi de ***gradul de codificare***, confundat uneori cu caracteristica de verticalitate-orizontalitate. Microinstrucţiunea cea mai simplă nu este codificată deloc, astfel încât fiecare bit controlează o singură resursă sau microoperaţie (figura 1.8*a*). Codificarea pe un singur nivel, sau directă, presupune ca biţii care controlează resurse mutual exclusive, cum ar fi UAL şi registrele de lucru dintr-o memorie locală, să fie combinaţi în câmpuri diferite (figura 1.8*b*). În codificarea pe două niveluri, sau indirectă, semnificaţia unui câmp depinde de valoarea altui câmp de control din microinstrucţiune (figura 1.8*c*). Acest tip de codificare este cunoscut şi sub numele de "bit steering". După cum se vede, câmpul C este decodificat, o parte din liniile de control mergând la decodificarea câmpului A , o parte la decodificarea câmpului B. În acest fel se poate mări puterea de control a unor câmpuri din microinstrucţiune. O altă variantă a acestei tehnici este prezentată în figura 1.8*d*. Aici acelaşi câmp controlează două resurse hardware cu funcţionare separată în timp. "Dirijarea" (steering) câmpului se face cu ajutorul unui bit de control. Într-un alt tip de codificare pe două niveluri, numită comutare de format, "format shifting", formatul microinstrucţiunii depinde de starea maşinii (de exemplu, pentru o unitate centrală, efectuarea unei operaţii de I/E sau prelucrarea unei instrucţiuni). Deşi codificarea microinstrucţiunii reduce volumul memoriei de microprograme, ea necesită timp pentru decodificare şi circuite suplimentare. SCM cu microprocesoare utilizează în general o codificare directă, pe un singur nivel.

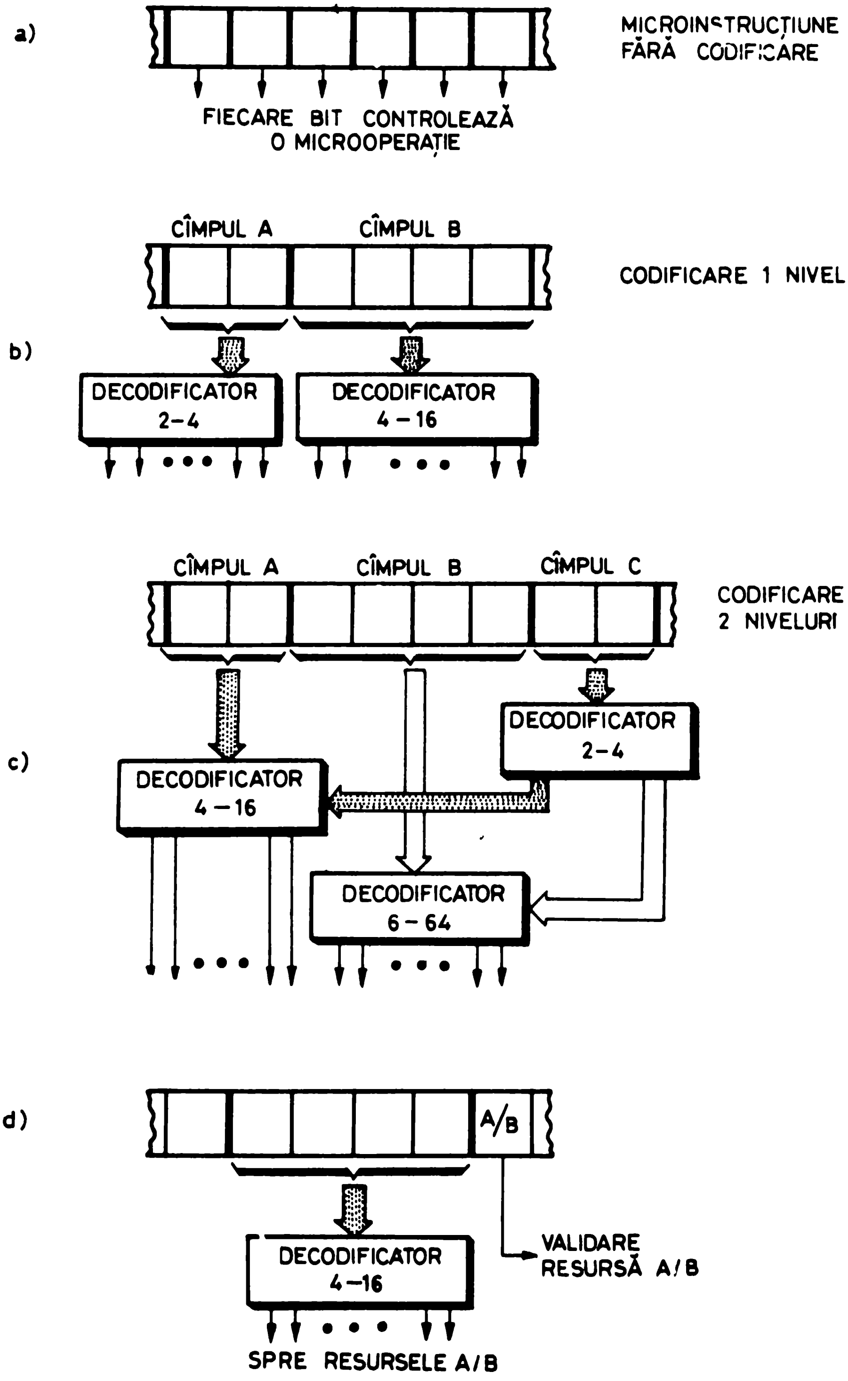


Fig. 1.8. Moduri de codificare a microinstrucțiunii

1.1.3.3. **Implementarea microinstrucţiunii**

Deşi execuţia microinstrucţiunilor se desfăşoară conform unei secvenţe citire-decodificare-execuţie, detaliile de implementare pot fi foarte diferite de la o soluţie la alta. Este vorba aici, în principal, despre caracteristicile serie-paralel şi monofază-polifază.

Caracteristica *serie-paralel* se referă la gradul de suprapunere între execuţia microinstrucţiunii curente şi citirea microinstrucţiunii următoare. Într-o implementare serie, citirea microinstrucţiunii următoare nu începe până nu se termină execuţia microinstrucţiunii curente. La cealaltă extremă, citirea microinstrucţiunii următoare este realizată în paralel cu execuţia microinstrucţiunii curente. Pentru o implementare serie (figura 1.9*a*), ciclul microinstrucţiunii, CMI, va fi suma dintre timpul de acces la memoria de microprograme, TA, şi timpul de execuţie, TE, al microoperaţiilor de control propriu-zis al resurselor hardware din maşină: CMI = TA + TE. În acest fel, durata unui microprogram de M microinstrucţiuni este de M · (TA + TE). Avantajul unei scheme seriale sau secvenţiale îl constituie simplitatea ei. O SCM secvenţială nu trebuie să controleze simultan execuţia şi citirea şi nu pune probleme în cazul microinstrucţiunilor de salt condiţionat. O SCM paralelă (figura 1.9*b*) sau de tip *pipeline*, cum este şi cea din figura 1.4, are avantajul unei viteze de lucru mai mari, execuţia microinstrucţiunii începând imediat după terminarea microinstrucţiunii precedente. În această implementare, ciclul microinstrucţiunii va fi dat de maximul dintre timpul de acces la memoria de microprograme şi timpul de execuţie: CMI = max (TA, TE). Durata minimă a unui microprogram de M microinstrucţiuni este în acest caz TA + M · TE. Ca dezavantaj, în afară de circuitele suplimentare, reprezentate în principal de registrul pentru memorarea microinstrucţiunii curente, registrul *pipeline*, la o SCM paralelă se pune şi problema salturilor condiţionate. Anticiparea adresei următoare se face pe baza unei estimări care să fie în general bună (de exemplu, la sfârşit de buclă se va presupune că bucla se reia - estimarea cea mai probabilă). Dacă anticiparea este corectă, majoritatea cazurilor, nu se va pierde din viteză. În cazul unei anticipări greşite se va pierde însă cel puţin un ciclu pentru citirea corectă a microinstrucţiunii ce urmează.

Caracteristica *monofază-polifază* se referă la numărul de faze (cicli secundari, subcicli) utilizate într-un ciclu principal, pentru execuţia unei microinstrucţiuni. Într-o implementare monofazată nu există cicli secundari ai ciclului de microinstrucţiune principal şi microinstrucţiunea este efectivă o singură perioadă de ceas, toate semnalele de control fiind generate simultan (figura 1.10*a*). Într-o implementare polifazată fiecare ciclu de ceas principal cuprinde mai mulţi cicli secundari. Semnalele de control sunt generate secvenţial corespunzător fiecărei faze (figura 1.10*b*). Controlul ciclului microinstrucţiunii se poate face cu ajutorul unui câmp care să fazeze în timp semnalele de control utilizate. În figura 1.10*c* este dată o altă variantă în care ciclul microinstrucţiunii este modificat de la o microinstrucţiune la alta în funcţie de timpul de execuţie. Acest ciclu variabil poate fi şi polifazat (figura 1.10*d*).

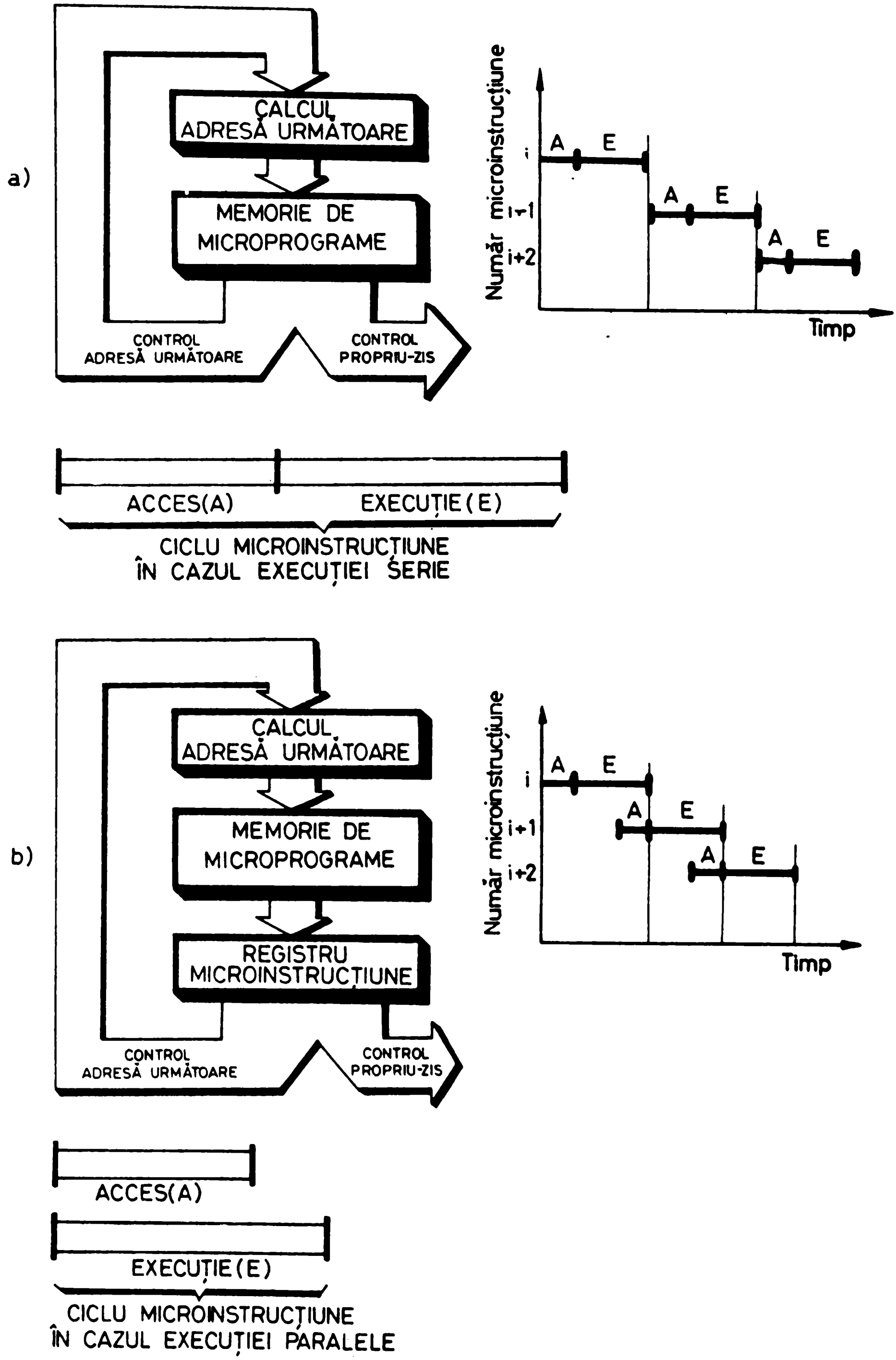


Fig. 1.9. Execuţia serială şi paralelă (în *pipeline*) a microinstrucţiunii

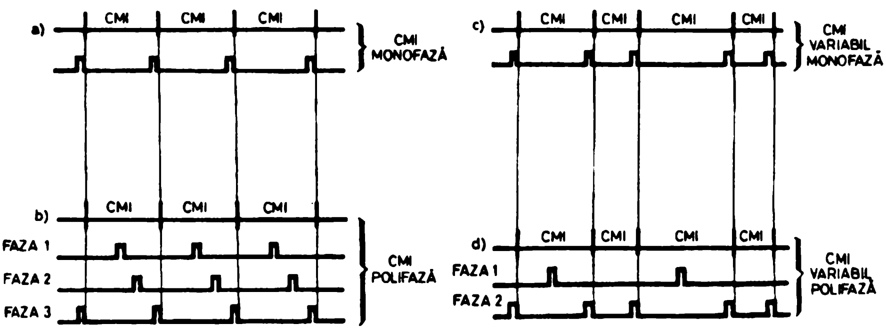


Fig. 1.10. Tipuri ale ciclului de microinstrucţiune (CMI)

1.1.4. AVANTAJELE ŞI DEZAVANTAJELE MICROPROGRAMĂRII

În raport cu metodele clasice, convenţionale, de proiectare, microprogramarea, ca modalitate de proiectare a structurilor numerice, prezintă următoarele avantaje:

1. *Flexibilitate, adaptabilitate.* Se ştie că una din caracteristicile pe care trebuie să le aibă structurile de control pentru a fi cât mai eficiente este flexibilitatea modului de control [15]. Această cerinţă poate fi asigurată de structurile de control programabile din rândul cărora fac parte şi SCM. Divizarea proiectului unei SCM într-un proiect de hardware şi un proiect de firmware permite ca microcodul să fie finalizat târziu în ciclul de proiectare, microprogramarea şi realizarea hardware a maşinii efectuându-se în paralel. Acest paralelism în proiectare oferă arhitecţilor de sistem posibilitatea să efectueze relativ comod modificări ulterioare de arhitectură. Prin modificarea microprogramelor (şi microprogramare dinamică) SCM se pot adapta uşor la noi interfeţe de proces.

Flexibilitatea SCM este dată şi de posibilitatea lor de a-şi reorganiza resursele hardware şi căile de circulaţie a informaţiei pentru a "traduce", interpreta, intrările în ieşiri şi functiuni, prin intermediul microprogramelor de control.

Tot aici trebuie vorbit şi despre *emulare*, care poate fi definită ca simularea unei maşini de către o alta prin folosirea tehnicilor de microprogramare. Astfel, o maşină a cărei funcţie de control este implementată cu SCM poate emula alte maşini, bineînţeles în cadrul limitativ al resurselor hardware pe care le are la dispoziţie. Noţiunea de emulare pune în evidenţă, faţă de interpretare, un aspect calitativ nou al maşinilor microprogramate: acela de a putea fi şi simulatoare sau, folosind termenul de mai sus, emulatoare. Acest aspect al microprogramării a apărut din necesitatea compatibilităţii sistemelor de calcul, mai ales din punct de vedere software. Calculatoarele microprogramate au putut fi echipate cu memorii de control suplimentare pentru emularea unor calculatoare mai vechi, eliminându-se astfel necesitatea reprogramării. Emulatorul va

reprezenta, în acest sens, "un set complet de microprograme care, atunci când este înscris în memoria de control, defineşte o maşină" [5]. Maşina pe care se face emularea este numită în general *maşină-gazdă* (host machine), iar maşina care este emulată, *maşină-ţintă* (target machine).

Microprogramarea poate deci extinde viaţa utilă a sistemului, care poate îndeplini noi cerinţe prin reprogramarea funcţiei de control. Acest lucru este adevărat pe timpul întregii vieţi a maşinii microprogramate. Posibilitatea de a răspunde unor cerinţe, precum şi posibilitatea de a extinde arhitectura unei maşini pentru a emula o altă maşină au fost avantaje deosebit de importante, îndeosebi pentru calculatoarele deja instalate, unde s-a pus problema eficientizării conversiei de programe şi de date.

2. *Uşurinţa în dezvoltare şi întreţinere.* Microprogramarea a fost definită şi ca o metodă sistematică de implementare a funcţiei de control pentru diferite maşini numerice. Aceasta reprezintă un avantaj evident faţă de metoda *ad-hoc* care caracterizează proiectarea convenţională a părţilor de control. În proiectarea convenţională efortul total este limitat în principiu de capacităţile proiectantului. Acesta are responsabilitatea trecerii de la arhitectura sistemului, definită de specificaţiile iniţiale, la organigramele de timp şi de funcţionare, precum şi responsabilitatea implementării acestor organigrame în hardware. Proiectantul logic are, de asemenea, responsabilitatea proiectării procedurilor de testare şi depanare a structurii de control, cea a realizării documentaţiei complete. Pe scurt, un proiect de structură de control convenţională nu poate fi privit decât ca un întreg care înglobează probleme de proiectare logică propriu-zisă, probleme de testare, întreţinere, documentare. Toate aceste probleme sunt dificile, complexe şi greu de rezolvat fără erori, mai ales în sistemele mari, tocmai datorită acestei concentrări. Pe de altă parte, microprogramarea oferă o divizare a proiectului, permiţând o îmbunătăţire a calităţii şi vitezei de realizare a produsului. Spre exemplu, în proiectarea unei maşini microprogramate se poate colabora cu un programator cu experienţă la proiectarea organigramelor şi scrierea microprogramelor. Cu ajutorul unui simulator software sau al unui sistem suport pentru microprogramare se pot verifica, depana şi compara diferite soluţii pentru a le optimiza din punctul de vedere al vitezei electronice, al volumului de microcod etc. La fel se poate produce şi un set clar, complet, standardizat, de documentaţie. De asemenea, microprogramarea oferă multe posibilităţi de automatizare a proiectării.

3. *Uniformitatea proiectării.* Legat de avantajul de mai sus, dar suficient de conturat şi important pentru a fi menţionat separat, este şi avantajul uniformităţii caracteristicilor de proiectare. Metoda microprogramării oferă ordine, uniformitate şi modularitate în proiectarea structurilor de control. Datorită acestei uniformităţi, care deseori înseamnă simplitate, se poate aprecia că este nevoie de mai puţine eforturi de învăţare şi experienţă pentru a forma microprogramatori pricepuţi decât proiectanţi de hardware convenţional.

4. *Eforturi de învăţare mai mici.* Datorită abordării simple, metodice şi organizate, microprogramarea este o tehnică de proiectare mai uşor de învăţat. De exemplu, pentru SCM funcţionarea poate fi înţeleasă numai cu ajutorul

organigramelor şi microprogramelor scrise simbolic, în timp ce pentru o structură de control realizată convenţional sunt necesare organigrame de timp şi de funcţionare, scheme logice complicate. Microprogramarea s-a dovedit a fi o metodă foarte utilă şi în alte probleme legate de ştiinţa calculatoarelor, cum ar fi cele referitoare la arhitectura şi organizarea sistemelor.

5. *Preţ de cost scăzut*. Un alt motiv pentru care microprogramarea a fost utilizată extensiv este şi preţul ei scăzut de implementare. Deşi economia de cost este dependentă şi de arhitectura structurii de control, de mijloacele de proiectare disponibile, SCM sunt ieftine în principal datorită dezvoltărilor tehnologice din domeniul memoriilor şi microprocesoarelor. Aceste circuite integrate pe scară largă reprezintă blocuri constructive ieftine cu care se pot implementa uşor şi eficient SCM care să controleze sisteme numerice de diverse complexităţi. Microprogramarea este o tehnică de proiectare care permite utilizarea circuitelor LSI şi deci construirea de structuri de control ieftine, puţin voluminoase, fiabile.

6. *Viteza*. Microprocesoarele *bit-slice*, realizate în tehnologii TTL Schottky sau ECL oferă, împreună cu memoriile semiconductoare foarte rapide, suportul hardware pentru implementarea unor aplicaţii ale microprogramării de viteză foarte mare. Pentru domenii cum ar fi, de exemplu, prelucrarea semnalelor, microprocesoarele MOS, tehnologiile convenţionale, sunt foarte lente şi/sau voluminoase, astfel încât structurile microprogramate cu microprocesoare *bit-slice* reprezintă una din cele mai rapide soluţii realizabile cu circuite LSI disponibile în comerţ. Deşi tehnologiile bipolare în care se realizează familiile de microprocesoare *bit-slice* nu permit o integrare foarte mare şi deşi funcţiile de control se implementează cu un volum mai mare de circuite, SCM realizate cu LSI rămân încă deosebit de eficiente pentru aplicaţii de mare viteză şi/sau specializate.

Un alt punct de vedere asupra vitezei este acela că în calculatoare implementarea microprogramată a unui algoritm s-a demostrat a fi de multe ori mai rapidă decât o implementare, echivalentă funcţional, într-un limbaj maşină.

Ca dezavantaje putem menţiona:

1. *Pierdere de performanţă*. Acest dezavantaj este principial şi se poate datora mai multor motive. În primul rând, el este un dezavantaj al tuturor structurilor de control programabile care, spre deosebire, de exemplu, de structurile analogice, implementează algoritmi de funcţionare secvenţial, pas cu pas, numeric. În al doilea rând există o pierdere de performanţă în SCM în comparaţie cu structurile realizate convenţional sau cu PLA-uri. Datorită puterii limitate a microinstrucţiunii pot apărea uneori întârzieri în prelucrare sau în adaptarea SCM la proces. Ceea ce se poate face printr-o logică convenţională, pur combinaţională, complicată, dar într-o singură perioadă de ceas, se realizează, de cele mai multe ori, în mai multe perioade de ceas, printr-o subrutină de microprogram. În al treilea rând, rezultă uneori o pierdere de timp, deci de performanţă, datorită imposibilităţii de a suprapune citirea microinstrucţiunii cu execuţia ei. Acest dezavantaj poate să apară în cazul SCM de tip *pipeline* numai pentru microinstrucţiuni de test sau permanent la SCM seriale.

2. *Volum, putere consumată mai mari.* La alegerea microprogramării ca metodă de implementare a unei structuri de control trebuie ţinut cont şi de faptul că SCM realizate cu circuite LSI bipolare sunt mai voluminoase şi consumă mai multă putere în comparaţie cu structurile de control realizate cu circuite integrate LSI MOS. Aceasta ca o particularizare a faptului că tehnologiile care oferă viteze mai ridicate (TTL Schottky, ECL) implică o putere consumată mai mare şi un grad de integrare mai mic.

Reevaluarea avantajelor şi dezavantajelor metodei microprogramării a condus în ultimul deceniu la o revenire la metoda convenţională prin arhitecturile de tip *RISC* (*Reduced Instruction Set Computer* - calculator cu set redus de instrucţiuni) şi prin circuitele de tip *ASIC* (*Application Specific Integrated Circuit* - circuit integrat specializat pe aplicaţie) pe care le vom prezenta în alte lucrări ale seriei.

1.2. MICROPROCESOARE *BIT-SLICE*

1.2.1. GENERALITĂŢI

În acest capitol vom descrie familiile cele mai cunoscute de microprocesoare *bit-slice*: Intel 3000, ca reprezentant al clasei microprocesoarelor *bit-slice* pe 2 biţi, şi Am 2900, ca reprezentant al microprocesoarelor *bit-slice* pe 4 biţi.

Menţionăm că familiile de circuite *bit-slice* sunt alcătuite din diverse blocuri constructive care au rolul de a permite implementarea cu uşurinţă a structurilor de control microprogramate. Aşa cum s-a spus în § 1.1, astfel de structuri au două funcţii principale – de secvenţiere şi de procesare propriu-zisă, de control – împărţite la rândul lor în următoarele subfuncţii:

– procesarea datelor;
– controlul adresei de microprogram;
– controlul adresei de macroprogram;
– controlul întreruperilor;
– accesul direct la memorie (DMA);
– controlul I/E;
– controlul memoriei.

Circuitele fiecărei familii trebuie să asigure implementarea comodă a subfuncţiilor. Din acest punct de vedere familia cea mai completă, cea mai diversificată, este 2900. Apreciem că diversitatea acestei familii a condus la deosebita popularitate de care ea se bucură în rândul proiectanţilor de structuri de control microprogramate. Familia cuprinde foarte multe circuite specializate destinate realizării unor subfuncţii de tipul celor enumerate mai sus, necesare într-o SCM. De exemplu, procesarea datelor poate fi asigurată de microprocesoarele *bit-slice* 2901, 2903, 29203 sau 29116, împreună cu generatoarele de transport anticipat de tipul 2902 şi circuitele 2904 pentru controlul operaţiilor de deplasare şi indicatorilor de condiţii. Controlul adresei de microprogram este asigurat de secvenţiatoarele de microprogram 2909/2911 sau 2910, iar controlul

adresei de macroprogram, de controlorul de program 2930. Controlul întreruperilor poate fi implementat cu ajutorul circuitelor de tipul 2914, accesul direct la memorie – cu circuitul 2940, I/E – cu *port*-urile 2950/2951, controlul memoriei cu controlorul de memorie dinamică 2964, iar cu circuitul 2960 detecţia şi corecţia erorilor. Alte familii asigură, în general, numai implementarea unor funcţii de bază ale SCM, cum sunt funcţia de secvenţiere şi/sau cea de procesare aritmetică.

TABELUL 1.1. **Familii de microprocesoare *bit-slice***

Familia (Fabricant principal)	Tehnologia	Frecvenţa de lucru maximă	Cuvântul cu care lucrează	Compatibilitatea TTL	Software de dezvoltare	Observaţii
2900 (AMD)	STTL	10 MHz	4 Biţi	Da	Da	Cea mai dezvoltată familie, cei mai mulţi furnizori
MACROLOGIC (FAIRCHILD)	STTL/ CMOS	10/2 MHz	4 Biţi	Da	Da	Versiune CMOS pentru aplicaţii de putere mică
F 100K (FAIRCHILD)	ECL	50 MHz	8 Biţi	Nu	Da	*Bit-slice* pe 8 biţi
3000 (INTEL)	STTL	10 MHz	2 Biţi	Da	Da	Prima familie de microprocesoare *bit-slice*, circuite pe 2 biţi
M 10800 (MOTOROLA)	ECL	20 MHz	4 Biţi	Nu	Da	Cea mai cunoscută familie de microprocesoare ECL
SBP-0400 (TEXAS INSTRUMENTS)	I^2L	5 MHz	4 Biţi	Da	Nu	Microprocesor *bit-slice* în tehnologie I^2L
74481/482 (TEXAS INSTRUMENTS)	STTL/ LSTTL	10 MHz	4 Biţi	Da	Da	Microprocesor *bit-slice* cu un set deosebit de bogat de instrucţiuni

În tabelul 1.1 se prezintă sintetic câteva familii de circuite *bit-slice* indicându-se tehnologia de fabricaţie, frecvenţa maximă de lucru, mărimea cuvântului, "feliei" procesate, compatibilitatea TTL, asigurarea cu mijloace software pentru dezvoltare.

1.2.2. SERIA INTEL 3000

1.2.2.1. Controlorul de microprogram Intel 3001

Intel 3001 este un circuit destinat implementării funcţiei de secvenţiere într-o structură de control microprogramată [17, 18].

Schema-bloc a circuitului este dată în figura 1.11.

Controlorul îndeplineşte în principal două tipuri de funcţiuni:

– funcţiuni de control al adresei microinstrucţiunii următoare:

– funcţiuni de control al indicatorilor de condiţie.

Corespunzător acestor funcţiuni, Intel 3001 conţine un registru-adresă-microprogram împreună cu logica de selecţie a adresei următoare, respectiv bistabilii C-flag, Z-flag şi *latch*-urile F.

Logica de selecţie a adresei următoare implementează un set de instrucţiuni de adresare condiţionate şi necondiţionate. Cu scopul minimizării numărului de ieşiri ale controlorului şi al reducerii logicii de selecţie a adresei următoare, Intel a proiectat circuitul 3001 pentru a adresa memorii de

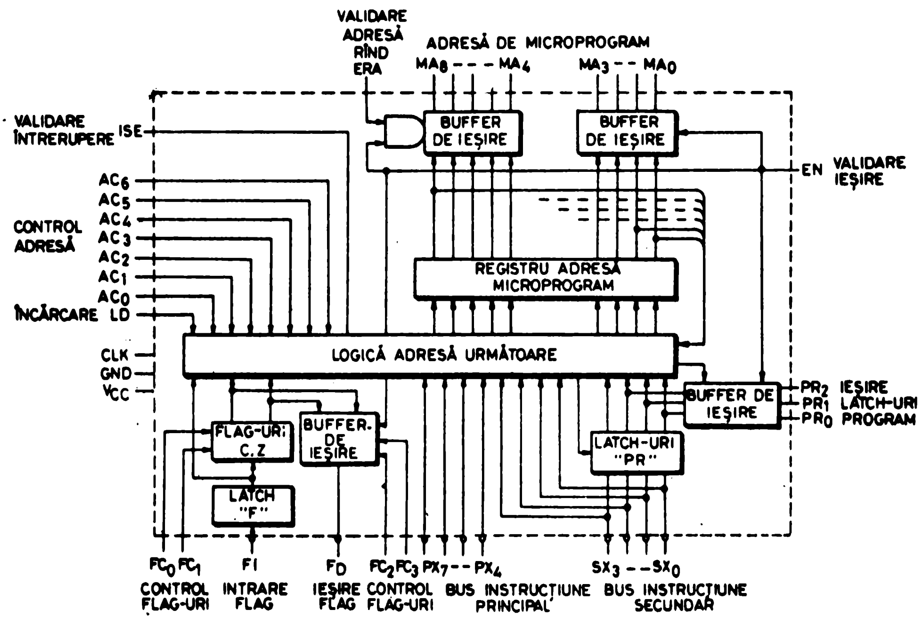


Fig. 1.11. Schema-bloc a controlorului de microprogram Intel 3001

microprogramare organizate matriceal. Fiecare adresă de microprogram este împărţită într-o adresă de rând şi o adresă de coloană. Un controlor Intel 3001 poate adresa, cu ajutorul celor 9 biţi de adresă MA_8-MA_0, maximum 512 microinstrucţiuni organizate într-o matrice de 32 rânduri şi 16 coloane. Logica de selecţie a adresei următoare este simplificată datorită folosirii acestei scheme

de adresare matriceală. De exemplu, pentru o anumită adresă de microinstrucţiune, microprogramul poate sări necondiţionat oriunde pe linia sau coloana respectivă, dar nu este posibil să sară oriunde în întreg spaţiul de memorie. Cu alte cuvinte microprogramul nu poate sări decât într-o anumită "zonă de salt" (*jump set*). Fiecare instrucţiune de adresare are o zonă de salt asociată.

Logica de condiţie, bistabilii C-flag, Z-flag şi *latch*-urile F permit implementarea unui set de funcţiuni care asigură salvarea valorii curente a ieşirii de transport a procesorului şi controlul intrării de transport în procesor. Aceste două tipuri de funcţiuni distincte sunt numite funcţiuni de intrare-condiţii şi funcţiuni de ieşire-condiţii. Logica de condiţie se poate utiliza împreună cu logica pentru deplasări şi transport din procesor, cu scopul implementării unor diverse funcţiuni aritmetice şi/sau de deplasare/rotire.

Instrucţiunile de adresare sunt selectate cu ajutorul intrărilor AC_0-AC_6. Adresa generată de logica de selecţie a adresei următoare este încărcată sincron pe frontul pozitiv al ceasului în registrul-adresă-microprogram. Adresa efectivă spre memoria de microprograme, MA_0-MA_8, este generată prin intermediul unor *driver*-e. Aşa cum s-a mai spus, adresa microinstrucţiunii următoare este împărţită în două părţi: adresa de rând, MA_8-MA_4, şi adresa de coloană, MA_3-MA_0.

Controlul indicatorilor de condiţie se face cu ajutorul intrărilor FC_0-FC_3.

Încărcarea registrului adresă microprogram cu informaţia de pe magistralele de instrucţiune PX_4-PX_7 şi SX_0-SX_3 se face pe frontul pozitiv al ceasului CLK atunci când intrarea de validare LD este pe "1". PX_4-PX_7 se încarcă în MA_0-MA_3, SX_0-SX_3 în MA_4-MA_7, iar MA_8 se poziţionează pe "0". Deci, se vede că informaţia de pe magistrala principală de instrucţiune PX_4-PX_7 poate specifica adresa coloanei, iar informaţia de pe magistrala secundară SX_0-SX_3, adresa rândului.

Validarea întreruperilor se face prin poziţionarea ieşirii ISE (*I*NTERRUPT *S*TROBE *E*NABLE). Ieşirea ISE este poziţionată pe "1" de instrucţiunea de adresare JZR în coloana 15. Microinstrucţiunea de pe rândul zero/coloana 15 poate reprezenta, de exemplu, începutul unui ciclu EXTRAGERE/EXECUŢIE atunci când controlorul 3001 este utilizat pentru implementarea microprogramată a unei unităţi centrale. Deci JZR în coloana 15, prin poziţionarea ieşirii ISE, poate valida un circuit de control al întreruperilor, de exemplu Intel 3214. Circuitul de control al întreruperilor răspunde la o întrerupere prin poziţionarea semnalului ERA (*E*NABLE *R*OW *A*DDRESS) pe zero, ceea ce va invalida *driver*-ele de ieşire pentru adresa de rând. În acest fel se permite generarea din exteriorul circuitului 3001 a adreselor de rând unde încep subrutinele de tratare a întreruperilor.

Funcţia de încărcare, cu ajutorul semnalului LD, este întotdeauna prioritară faţă de controlul adresei prin intermediul intrărilor AC_0-AC_6. Totuşi această încărcare nu este mai prioritară decât citirea sau încărcarea *latch*-urilor PR cu ajutorul instrucţiunilor JCE, respectiv JPX. De asemenea, încărcarea registrului adresă microprogram nu perturbă controlul întreruperilor, poziţionarea ISE şi controlul indicatorilor de condiţie.

1.2.2.2. **Microprocesorul Intel 3002**

Intel 3002 este un microprocesor *bit-slice* de 2 biţi conectabil în cascadă. Schema-bloc a circuitului este dată în figura 1.12.

După cum se vede, Intel 3002 este format din următoarele elemente principale:

– decodificatorul de microfuncţiuni;
– unitatea aritmetică-logică, UAL, şi multiplexoarele A, B;
– registrele de lucru R_0-R_9 şi T;
– registrul acumulator AC;
– registrul-adresă-memorie MAR.

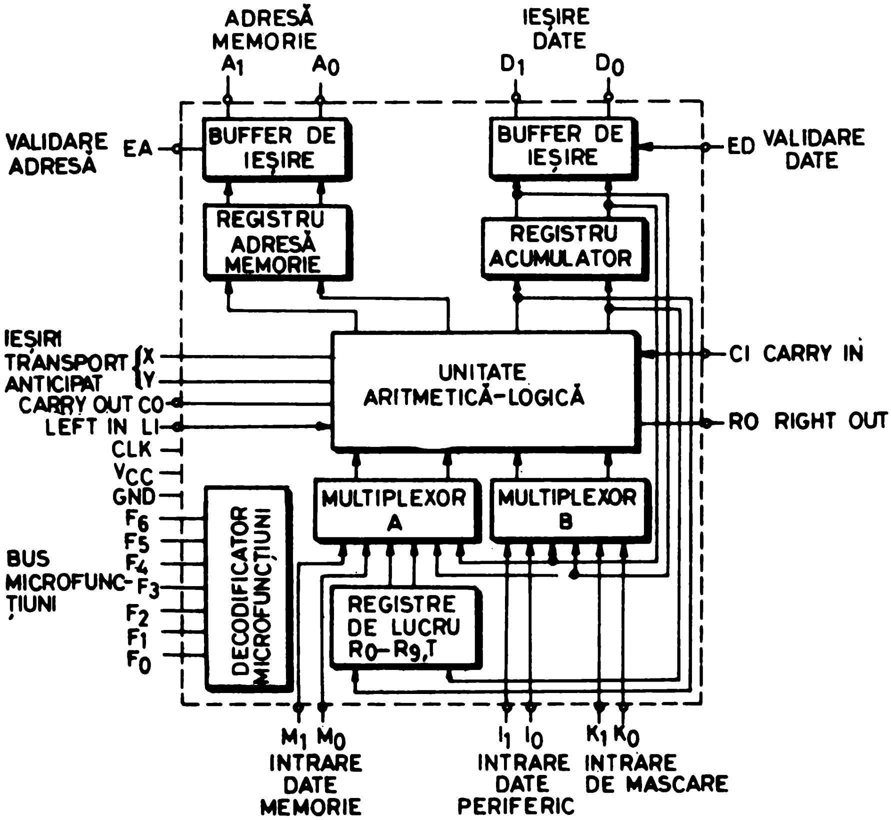


Fig. 1.12. Schema-bloc a microprocesorului *bit-slice* Intel 3002

Decodificatorul de microfuncţiuni decodifică intrările F_0-F_6 pentru a genera semnalele de selecţie a operaţiei UAL, adresa registrului de lucru şi semnalele de control pentru multiplexoarele A şi B.

Unitatea aritmetică-logică efectuează operaţii aritmetice şi logice. Rezultatul unei operaţii UAL poate fi încărcat în registrul acumulator sau într-unul din registrele de lucru. Intrarea LI (LEFT IN) şi ieşirea RO (RIGHT

OUT) sunt utilizate în operaţiile de deplasare dreapta. Intrarea şi ieşirea de transport, CI, respectiv CO, servesc pentru cuplarea circuitului într-o structură lentă cu transport-serie. Pentru o structură rapidă cu transport anticipat circuitul generează semnale standard X şi Y. Aceste semnale pot fi utilizate de generatorul de transport anticipat Intel 3003. Multiplexoarele A şi B selectează cele două intrări în UAL. Multiplexorul A are ca intrări magistrala M, registrele de lucru şi acumulatorul, iar multiplexorul B, magistrala I, magistrala K şi acumulatorul. Magistrala K maschează întotdeauna celelalte intrări selectate de multiplexorul B, asigurându-se în acest fel un mijloc comod de mascare şi testare la nivel de bit.

Registrele de lucru R_0-R_9 şi T sunt încărcate cu ieşirea UAL. Ieşirile registrelor de lucru sunt multiplexate pentru a putea fi selectate ca intrare în UAL.

Registrul acumulator se încarcă cu rezultatul unei operaţii UAL. Acumulatorul poate fi selectat ca operand UAL sau citit pe magistrala de date D. De obicei magistrala D este utilizată pentru transmiterea informaţiei spre memoria principală sau spre dispozitivele de I/E.

Registrul-adresă-memorie se încarcă cu rezultatul unei operaţii UAL şi poate fi citit pe magistrala de adresă A. MAR şi magistrala A pot fi utilizate pentru adresarea memoriei principale sau pentru selectarea unui dispozitiv de I/E.

1.2.3. FAMILIA Am 2900

1.2.3.1. **Microprocesorul *bit-slice* Am 2901**

Acest microprocesor *bit-slice* de 4 biţi este conceput ca bloc constructiv de mare viteză şi este destinat proiectării de unităţi centrale, de dispozitive de control al I/E, în general, implementării de structuri de control microprogramate utilizabile în numeroase aplicaţii. Flexibilitatea microprogramării acestui circuit a permis emularea eficientă a multor maşini de calcul digitale [16].

Microprocesorul, aşa cum se vede în schema bloc din figura 1.13, este compus din următoarele elemente principale:

- o memorie RAM de 16 x 4 biţi cu două *port*-uri de ieşire;
- o unitate aritmetică-logică de mare viteză;
- un registru auxiliar Q;
- circuite pentru deplasări, decodificări şi multiplexări.

Cuvântul de comandă al microprocesorului, microinstrucţiunea, are nouă biţi organizaţi în trei câmpuri de câte trei biţi fiecare. Aceste câmpuri selectează operanzii-sursă pentru UAL, funcţia UAL şi destinaţia rezultatului UAL. Microprocesorul are ieşiri "trei-stări", generează diverse semnale de stare din UAL şi poate fi conectat în cascadă cu transport-serie sau anticipat.

În figura 1.14 se dă schema detaliată a microprocesorului Am 2901.

Memoria RAM cu două *port*-uri de ieşire permite ca oricare din cele 16

cuvinte să poată fi citit la *port*-ul A, selectat de câmpul de adresă A de 4 biţi, şi la *port*-ul B, selectat de câmpul de adresă B, de asemenea de 4 biţi. Dacă cele două câmpuri de selecţie sunt identice, atunci la ieşirile celor două *port*-uri va fi citit acelaşi cuvânt din RAM.

Scrierea în RAM se face prin validarea semnalului de scriere (RAMEN), adresa cuvântului ce va fi codificat fiind dată de valoarea câmpului de adresă B. Informaţia de intrare în RAM este generată prin intermediul unor multiplexoare cu trei intrări. Aceste multiplexoare permit ca ieşirea UAL, F, să fie deplasată stânga sau dreapta cu o poziţie ori să intre nemodificată în RAM.

Ieşirile RAM-ului sunt înscrise în *latch*-urile A şi B, fiecare de câte 4 biţi. Aceste *latch*-uri păstrează informaţia de la ieşirile RAM-ului pe timpul cât ceasul CP este "0".

În acest fel se elimină eventualele instabilităţi la scrierea unei noi informaţii în RAM.

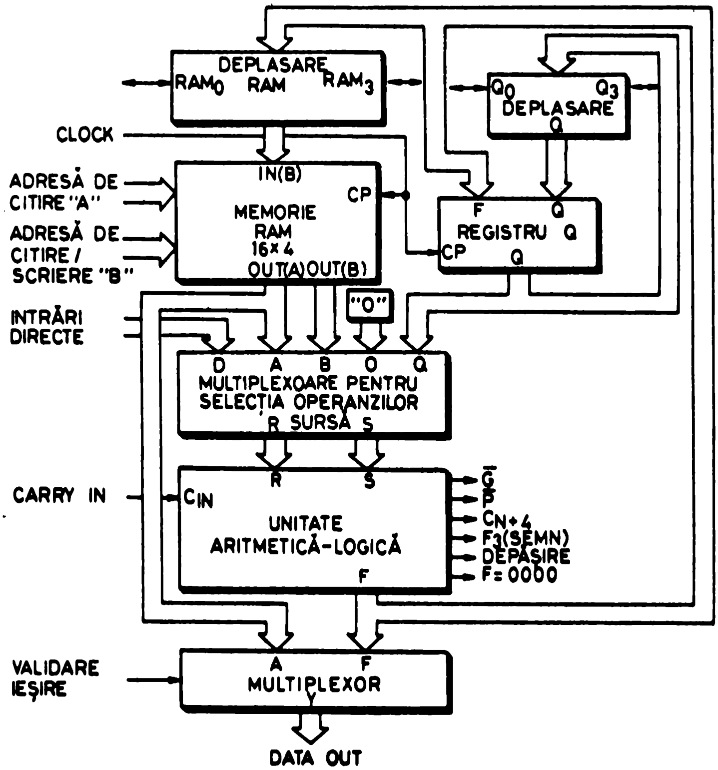


Fig. 1.13. Schema-bloc a microprocesorului *bit-slice* Am 2901

UAL din Am 2901 poate realiza asupra celor două intrări R şi S, de câte 4 biţi, trei operaţii aritmetice binare şi cinci operaţii logice. Intrarea R este obţinută prin intermediul unor multiplexoare cu două intrări, iar intrarea S, cu

ajutorul unor multiplexoare cu trei intrări. Multiplexoarele pot fi inhibate, ceea ce echivalează cu operanzi ZERO. De menţionat că multiplexoarele R. au ca intrări *port*-ul A al RAM-ului şi intrarea directă D, iar multiplexoarele S, *port*-urile A şi B ale RAM-ului şi registrul Q.

Utilizarea multiplexoarelor la intrările UAL permite selecţia ca operanzi-sursă a opt perechi formate cu intrările A, B, D, Q şi "0". Selecţia operanzilor-sursă UAL se face cu ajutorul câmpului I_0-I_2 din microinstrucţiune. Valorile acestui câmp pentru cele opt perechi de operanzi-sursă sunt date în tabelul 1.2.

Intrarea D reprezintă o intrare directă de informaţie, utilizată pentru a introduce noi valori în registrele de lucru sau, prin intermediul UAL, pentru a modifica valorile acestor registre. Registrul Q este un registru, tot de 4 biţi, destinat în principal implementării rutinelor de înmulţire şi împărţire. El poate fi însă utilizat şi ca registru acumulator sau de memorare în cazul altor aplicaţii.

Unitatea aritmetică-logică UAL este un dispozitiv aritmetic şi logic de mare viteză ale cărui funcţii sunt selectate de câmpul I_3-I_5 din microinstrucţiune. Valorile acestui câmp sunt date în tabelul 1.3. Ieşirile $\overline{G}$ şi $\overline{P}$, de generare-transport şi, respectiv, de propagare-transport, se utilizează ca intrări în generatorul de transport anticipat Am 2902, în cazul unei conectări în cascadă. Ieşirea C_{n+4}, transportul, poate fi utilizată ca indicator de depăşire într-un registru de stare-program.

UAL mai are încă trei ieşiri, F3, F=0, OVR, ce pot fi utilizate ca indicatori de condiţie. Ieşirea F3 reprezintă cel mai semnificativ bit al UAL, semnul. Ea poate fi utilizată pentru a determina semnul rezultatului fără a valida ieşirile "trei-stări". Ieşirea F=0 este întrebuinţată pentru detecţia de zero. Este o ieşire de tip "colector în gol" care permite cablarea unui SAU LOGIC între mai multe circuite Am 2901. Ieşirea OVR este folosită pentru a indica, în cazul operaţiilor aritmetice, depăşirea gamei de numere reprezentate în complement faţă de doi.

Ieşirile F_0-F_3 ale UAL pot avea diverse destinaţii selectate de câmpul I_6-I_8 din microinstrucţiune. Valorile acestui câmp sunt date în tabelul 1.4. Ieşirile Y_0-Y_3 ale microprocesorului sunt de tip "trei-stări" şi se pot conecta direct în sistem magistrală. Multiplexorul cu două intrări care generează aceste ieşiri poate selecta *port*-ul A al RAM-ului sau ieşirile F ale UAL. Această selectare este controlată, de asemenea, de câmpul I_6-I_8.

Câmpul I_6-I_8 controlează şi operaţiile de deplasare. Aşa cum s-a mai spus, intrarea în RAM se face prin intermediul unor multiplexoare cu trei intrări. Acestea permit ca ieşirile UAL să intre în RAM deplasate dreapta sau stânga. Deplasarea se face prin intermediul a două intrări/ieşiri numite RAM_0 şi RAM_3. Astfel, la deplasare stânga se validează ieşirea RAM_3 şi intrarea RAM_0, la deplasare dreapta se validează ieşirea RAM_0 şi intrarea RAM_3, iar când nu se face nici o deplasare intrările/ieşirile RAM_0, RAM_3 sunt invalidate. Câmpul I_6-I_8 controlează, de asemenea, şi unitatea de deplasare asociată registrului Q. Ca şi deplasarea RAM-ului deplasarea registrului Q se face prin intermediul unui multiplexor şi a unor intrări/ieşiri notate Q_0 şi Q_3.

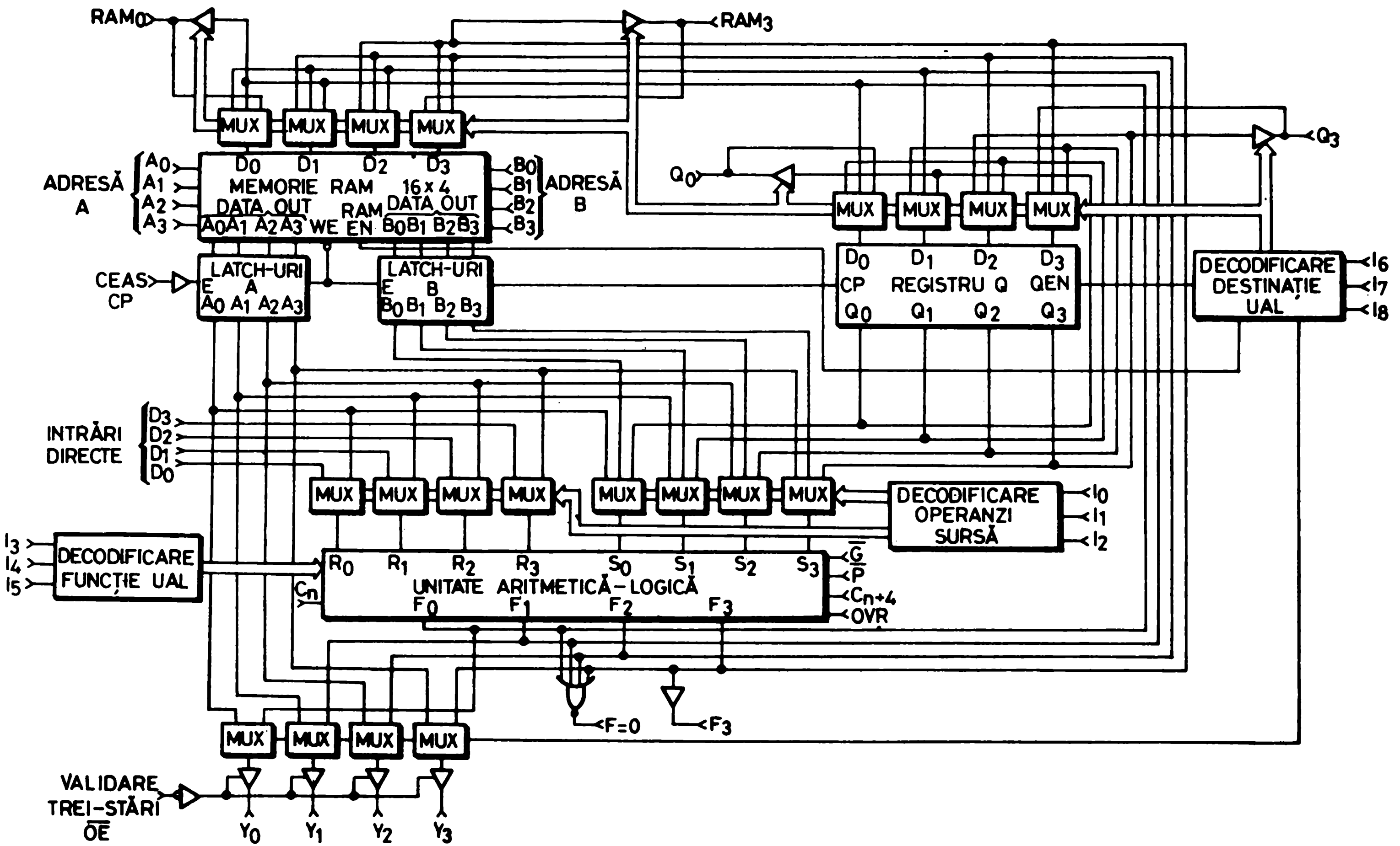


Fig. 1.14. Schema detaliată a microprocesorului Am 2901

TABELUL 1.2. Selecția operanzilor

Cod mnemonic	I_2	I_1	I_0	Operanzi UAL (R, S)	
AQ	0	0	0	A	Q
AB	0	0	1	A	B
ZQ	0	1	0	0	Q
ZB	0	1	1	0	B
ZA	1	0	0	0	A
DA	1	0	1	D	A
DQ	1	1	0	D	Q
DZ	1	1	1	D	0

TABELUL 1.3. Funcțiile UAL

Cod mnemonic	I_5	I_4	I_3	Funcția UAL	Simbol
ADD	0	0	0	R Plus S	$R + S$
SUBR	0	0	1	S Minus R	$S - R$
SUBS	0	1	0	R Minus S	$R - S$
OR	0	1	1	R OR S	$R \vee S$
AND	1	0	0	R AND S	$R \wedge S$
NOTRS	1	0	1	$\overline{R}$ AND S	$\overline{R} \wedge S$
EXOR	1	1	0	R EX-OR S	$R \veebar S$
EXNOR	1	1	1	R EX-NOR S	$\overline{R \veebar S}$

TABELUL 1.4. Destinațiile UAL

Cod mnemonic	I_8	I_7	I_6	Funcția RAM		Funcția Q		Ieșirea Y	Deplasare RAM		Deplasare Q	
				deplasare	încărcare	deplasare	încărcare		RAM_0	RAM_3	Q_0	Q_3
QREG	0	0	0	X	Nu	Nu	F→Q	F	X	X	X	X
NOP	0	0	1	X	Nu	X	Nu	F	X	X	X	X
RAMA	0	1	0	Nu	F→B	X	Nu	A	X	X	X	X
RAMF	0	1	1	Nu	F→B	X	Nu	F	X	X	X	X
RAMQD	1	0	0	Dreapta	F/2→B	Dreapta	Q/2→Q	F	F_0	IN_3	Q_0	IN_3
RAMD	1	0	1	Dreapta	F/2→B	X	Nu	F	F_0	IN_3	Q_0	X
RAMQU	1	1	0	Stânga	2F→B	Stânga	2Q→Q	F	IN_0	F_3	IN_0	Q_3
RAMU	1	1	1	Stânga	2F→B	X	Nu	F	IN_0	F_3	X	Q_3

Notă: X – fără importanță, intrarea de deplasare este conectată la o ieșire internă în starea a treia; B – registru RAM adresat de intrările B; IN – intrare.

TABELUL 1.5. Operațiile executate de UAL

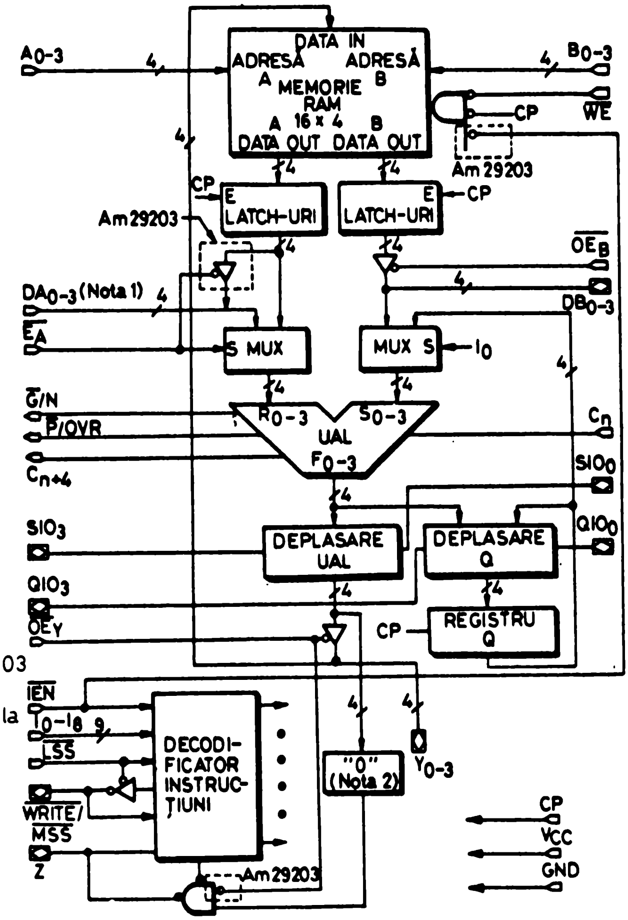
I_{543}	Funcția UAL	I_{210}							
		0	1	2	3	4	5	6	7
		Sursa UAL							
		A, Q	A, B	0, Q	0, B	0, A	D, A	D, Q	D, 0
0	$C_n = 0$ R Plus S $C_n = 1$	A+Q A+Q+1	A+B A+B+1	Q Q+1	B B+1	A A+1	D+A D+A+1	D+Q D+Q+1	D D+1
1	$C_n = 0$ S Minus R $C_n = 1$	Q−A−1 Q−A	B−A−1 B−A	Q−1 Q	B−1 B	A−1 A	A−D−1 A−D	Q−D−1 Q−D	−D−1 −D
2	$C_n = 0$ R Minus S $C_n = 1$	A−Q−1 A−Q	A−B−1 A−B	−Q−1 −Q	−B−1 −B	−A−1 −A	D−A−1 D−A	D−Q−1 D−Q	D−1 D
3	R OR S	$A \vee Q$	$A \vee B$	Q	B	A	$D \vee A$	$D \vee Q$	D
4	R AND S	$A \wedge Q$	$A \wedge B$	0	0	0	$D \wedge A$	$D \wedge Q$	0
5	$\overline{R}$ AND S	$\overline{A} \wedge Q$	$\overline{A} \wedge B$	Q	B	A	$\overline{D} \wedge A$	$\overline{D} \wedge Q$	0
6	R EX-OR S	$A \veebar Q$	$A \veebar B$	Q	B	A	$D \veebar A$	$D \veebar Q$	D
7	R EX-NOR S	$\overline{A \veebar Q}$	$\overline{A \veebar B}$	$\overline{Q}$	$\overline{B}$	$\overline{A}$	$\overline{D \veebar A}$	$\overline{D \veebar A}$	$\overline{D}$

În tabelul 1.5 se sintetizează operaţiile executate de UAL asupra tuturor perechilor de operanzi ce pot fi selectate cu ajutorul câmpului I_0-I_2. Este precizat, de asemenea, şi rolul pe care îl are intrarea C_n asupra operaţiilor aritmetice. Matricea din această figură defineşte complet funcţionarea UAL împreună cu selecţia corespunzătoare a operanzilor.

1.2.3.2. Microprocesoarele Am 2903/29203

Microprocesorul *bit-slice* Am 2903 este un microprocesor pe patru biţi expandabil. El realizează toate funcţiunile lui Am 2901, având în plus unele îmbunătăţiri destinate în special implementării de procesoare aritmetice. Am 29203, similar cu Am 2903, are în plus faţă de acesta câteva facilităţi de I/E, un set de instrucţiuni mai bogat şi este cu aproximativ 30% mai rapid.

În figura 1.15 se dă schema-bloc a microprocesoarelor Am 2903/ 29203.



1. DA0-3 este intrare pentru Am 2903 şi I/E pentru Am 29203
2. La Am 2903 "0" este conectat la ieşirile Y după buffer-ul $\overline{OE_Y}$

Fig. 1.15. Schema-bloc a microprocesoarelor Am 2903/29203

Aceste circuite constau în principal dintr-un RAM de 16 x 4 biţi cu două *port*-uri de ieşire şi *latch*-urile corespunzătoare, o UAL şi o unitate de deplasare asociată ieşirii UAL, un registru Q împreună cu unitatea pentru deplasare asociată lui şi un decodificator de instrucţiuni. Cuvântul de comandă al microprocesorului are 9 biţi, $I_0 - I_8$.

Memoria RAM permite citirea simultană la *port*-urile de ieşire a oricăror două locaţii adresate de câmpurile A şi B. Ca la Am 2901, la cele două ieşiri pot fi citite informaţii identice dacă cele două adrese, A şi B, sunt egale. *Latch*-urile, cu acelaşi rol ca la Am 2901, sunt transparente atunci când ceasul CP este "1" şi memorează informaţia de la ieşirea memoriei RAM, când ceasul este "0". Ieşirea RAM poate fi citită în afara circuitului la *port*-ul de I/E DB pentru Am 2903 şi/sau la *port*-ul DA, pentru Am 29203. Operaţiile de citire la aceste *port*-uri sunt validate de semnalele $\overline{OE}_B$, respectiv $\overline{E}_A$.

Port-ul de I/E Y serveşte ca intrare pentru scrierea de informaţii externe în RAM şi ca ieşire pentru citirea în exteriorul microprocesorului a ieşirii UAL.

Scrierea în RAM se face la adresa B, atunci când semnalul de validare $\overline{WE}$ şi ceasul CP sunt "0".

UAL poate realiza şapte operaţii aritmetice şi nouă operaţii logice. Multiplexoarele de la intrările UAL permit selectarea a numeroase perechi de operanzi. Astfel, operandul R se obţine prin intermediul unui multiplexor cu ajutorul semnalului $\overline{E}_A$, care selectează fie intrarea externă DA, fie *port*-ul A al RAM-ului. Operandul S, obţinut de asemenea prin intermediul unui multiplexor, poate fi *port*-ul B al RAM-ului, intrarea externă DB sau registrul Q. Selecţia se face cu ajutorul semnalelor $\overline{OE}_B$ şi I_0. În tabelul 1.6 sunt date toate perechile de operanzi UAL în funcţie de semnalele de selecţie $\overline{E}_A$, I_0, $\overline{OE}_B$.

TABELUL 1.6. **Operanzii UAL**

$\overline{E}_A$	I_0	$\overline{OE}_B$	Operand R	Operand S
0	0	0	Ieşirea "A" a RAM	Ieşirea "B" a RAM
0	0	1	Ieşirea "A" a RAM	DB_{0-3}
0	1	X	Ieşirea "A" a RAM	Registrul Q
1	0	0	DA_{0-3}	Ieşirea "B" a RAM
1	0	1	DA_{0-3}	DB_{0-3}
1	1	X	DA_{0-3}	Registrul Q

Funcţiile realizate de UAL din Am 2903 sunt selectate cu ajutorul biţilor de instrucţiune $I_8 - I_0$ (tabelele 1.7 şi 1.8). Atunci când I_4, I_3, I_2, I_1 şi I_0 sunt "0"

Am 2903 execută funcţii speciale (tabelul 1.8) selectate cu ajutorul biţilor $I_8 - I_5$. Când Am 2903 nu efectuează funcţii speciale atunci operaţia UAL este determinată de valoarea biţilor $I_4 - I_1$ (tabelul 1.7). UAL din Am 29203 este similară, cu singura diferenţă că ea execută 16 funcţii speciale în loc de 9 câte execută cea din Am 2903.

De remarcat că Am 2903/29203 pot fi conectate în cascadă cu transport-serie sau anticipat. Într-o conectare în cascadă fiecare microprocesor trebuie programat în funcţie de poziţia lui: cel mai semnificativ, intermediar sau cel mai puţin semnificativ. La conectarea în cascadă se folosesc semnalele $\overline{G}$ şi $\overline{P}$ generate de circuitul cel mai puţin semnificativ şi de circuitele intermediare.

TABELUL 1.7. **Funcţiile UAL**

I_4	I_3	I_2	I_1	I_0	Funcţii UAL
0	0	0	0	0	Funcţii speciale (v. tab. 1.8.)
0	0	0	0	1	$F_i = 1$
0	0	0	1	X	F = S Minus R Minus 1 Plus C_n
0	0	1	0	X	F = R Minus S Minus 1 Plus C_n
0	0	1	1	X	F = R Plus S Plus C_n
0	1	0	0	X	F = S Plus C_n
0	1	0	1	X	F = $\overline{S}$ Plus C_n
0	1	1	0	0	Funcţii speciale rezervate
0	1	1	0	1	F = R Plus C_n
0	1	1	1	0	Funcţii speciale rezervate
0	1	1	1	1	F = $\overline{R}$ Plus C_n
1	0	0	0	0	Funcţii speciale rezervate
1	0	0	0	1	$F_i = 0$
1	0	0	1	X	$F_i = R_i$ AND S_i
1	0	1	0	X	$F_i = R_i$ EXCLUSIVE NOR S_i
1	0	1	1	X	$F_i = R_i$ EXCLUSIVE OR S_i
1	1	0	0	X	$F_i = R_i$ AND S_i
1	1	0	1	X	$F_i = R_i$ NOR S_i
1	1	1	0	X	$F_i = R_i$ NAND S_i
1	1	1	1	X	$F_i = R_i$ OR S_i

În afara semnalelor de transport anticipat $\overline{G}$ şi $\overline{P}$ microprocesoarele Am 2903/29203 mai generează: semnalul de depăşire binară C_{n+4}, indicatorul N (cel mai semnificativ bit al ieşirii UAL), care se poate utiliza pentru determinarea semnului, semnalul de depăşire OVR utilizat pentru a indica operaţiile aritmetice

care depăşesc gama de numere reprezentate în complement faţă de doi. Pentru generarea semnalelor $\overline{G}$, $\overline{P}$, N, OVR se folosesc numai două ieşiri, astfel încât semnalele $\overline{G}$ şi $\overline{P}$ sunt obţinute la ieşirile celui mai puţin semnificativ circuit şi ale circuitelor intermediare, iar semnalele N şi OVR – la ieşirile celui mai semnificativ circuit.

Microprocesoarele Am 2903/29203 au prevăzută o intrare/ieşire de zero. Această intrare/ieşire, Z, de tip "colector în gol", poate fi conectată într-un SAU LOGIC între mai multe circuite. Ca ieşire se poate utiliza drept indicator de zero, semnalând situaţia în care pinii Y_0-Y_3 sunt "0". Pentru Am 29203 ieşirea Z poate fi "1" numai dacă semnalul de validare $\overline{OE_Y}$ este "1". În acest fel detecţia de zero se poate face pe mai puţin de un cuvânt.

În tabelul 1.9 sunt date ieşirile UAL, Y_0-Y_3, în funcţie de instrucţiunea executată.

Unitatea de deplasare UAL permite trecerea nemodificată a ieşirii F a UAL, deplasarea stânga cu o poziţie (2F) şi deplasarea dreapta cu o poziţie (F/2). Sunt posibile atât deplasări aritmetice, cât şi deplasări logice. Deplasarea se face, ca şi la Am 2901, prin intermediul a două intrări/ieşiri notate SIO_0 şi SIO_3. În timpul unei deplasări stânga SIO_0 este validată ca intrare, iar SIO_3 ca ieşire. În timpul unei deplasări dreapta SIO_3 este validată ca intrare şi SIO_0 ca ieşire. În tabelul 1.9 se dau şi semnificaţiile semnalelor SIO_0 şi SIO_3 în funcţie de instrucţiunea executată.

În unitatea de deplasare există şi un generator/controlor de paritate de cinci biţi care permite detecţia erorilor la ieşirea UAL. Paritatea poate fi generată, sub controlul instrucţiunii, la ieşirea SIO_0 pentru F_0, F_1, F_2, F_3, SIO_3. În tabelele 1.8 şi 1.9 sunt definite toate operaţiile executate de unitatea de deplasare asociată UAL.

Registrul auxiliar Q este destinat, ca şi la Am 2901, în principal implementării rutinelor de înmulţire şi împărţire putând fi însă utilizat, în unele aplicaţii, şi ca registru acumulator sau de memorare. Acest registru poate fi selectat ca operand-sursă pentru UAL şi încărcat cu ieşirea F a UAL. Unitatea de deplasare asociată registrului Q asigură numai deplasări logice stânga (2Q) sau dreapta (Q/2). Intrările/ieşirile folosite pentru aceste deplasări sunt QIO_0 şi QIO_3.

Am 2903/29203 permit executarea operaţiilor de deplasare logice şi aritmetice pe dublu-cuvânt, prin conectarea ieşirii QIO_3 a celui mai semnificativ circuit la intrarea SIO_0 a celui mai puţin semnificativ şi prin executarea unei instrucţiuni de deplasare atât a ieşirii UAL, cât şi a registrului Q. Operaţiile executate de registrul Q şi de unitatea de deplasare asociată lui, în funcţie de biţii de instrucţiune I_8-I_5, sunt date, de asemenea, în tabelele 1.8 şi 1.9.

Semnalele de control interne sunt generate de un decodificator de instrucţiuni în funcţie de: intrările de instrucţiune I_0-I_8, intrarea de validare-instrucţiune $\overline{IEN}$, intrarea $\overline{LSS}$ şi intrarea/ieşirea $\overline{WRITE}/\overline{MSS}$

TABELUL 1.8. **Funcțiile speciale realizate de Am 2903/29203**

I_8	I_7	I_6	I_5	Funcția specială	Funcția UAL	Deplasarea UAL	Deplasarea Q	Circuitul care implementează funcția
0	0	0	0	Înmulțire fără semn	$F=S+C_n$ dacă $Z=0$ $F=R+S+C_n$ dacă $Z=1$	Deplasare logică $F/2 \to Y$ (Nota 1)	Deplasare logică $Q/2 \to Q$	Am 2903 Am 29203
0	0	0	1					Am 29203
0	0	1	0	Înmulțire în complement față de 2	$F=S+C_n$ dacă $Z=0$ $F=R+S+C_n$ dacă $Z=1$	Deplasare logică $F/2 \to Y$ (Nota 2)	Deplasare logică $Q/2 \to Q$	Am 2903 Am 29203
0	0	1	1					Am 29203
0	1	0	0	Incrementare cu 1 sau 2	$F=S+1+C_n$	$F \to Y$	Nu	Am 2903 Am 29203
0	1	0	1	Semn-mărime/complement față de 2	$F=S+C_n$ dacă $Z=0$ $F=\overline{S}+C_n$ dacă $Z=0$	$F \to Y$ (Nota 3)	Nu	Am 2903 Am 29203
0	1	1	0	Înmulțire în complement față de 2, ultimul ciclu	$F=S+C_n$ dacă $Z=1$ $F=S-R-1+C_n$ dacă $Z=1$	$F/2 \to Y$ (Nota 2)	Deplasare logică $Q/2 \to Q$	Am 2903 Am 29203
0	1	1	1					Am 29203
1	0	0	0	Normalizare precizie simplă	$F=S+C_n$	$F \to Y$	Deplasare logică $2Q \to Q$	Am 2903 Am 29203
1	0	0	1					Am 29203
1	0	1	0	Normalizare precizie dublă și împărțire, primul ciclu	$F=S+C_n$	Deplasare logică $2F \to Y$	Deplasare logică $2Q \to Q$	Am 2903 Am 29203
1	0	1	1					Am 29203
1	1	0	0	Împărțire în complement față de 2	$F=S+R+C_n$ dacă $Z=0$ $F=S-R-1+C_n$ dacă $Z=1$	Deplasare logică $2F \to Y$	Deplasare logică $2Q \to Q$	Am 2903 Am 29203
1	1	0	1					Am 29203
1	1	1	0	Împărțire în complement față de 2, corecție și rest	$F=S+R+C_n$ dacă $Z=0$ $F=S-R-1+C_n$ dacă $Z=1$	$F \to Y$	Deplasare logică $2Q \to Q$	Am 2903 Am 29203
1	1	1	1					Am 29203

Note: 1. La cel mai semnificativ circuit C_{n+4} este conectat intern la ieșirea Y_3.
2. La cel mai semnificativ circuit $F_3 \veebar OVR$ este conectat intern la ieșirea Y_3.
3. La ieșirea Y_3 a celui mai semnificativ circuit se generează $S_3 \veebar F_3$.

TABELUL 1.9. **Ieşirile UAL în funcţie de instrucţiunea executată**

I_8	I_7	I_6	I_5	Deplasare UAL	SIO_3		Y_3		Y_2		Y_1	Y_0	SIO_0	Deplasare Q	QIO_3	QIO_0
					C.M.S. circuit	alte circuite	C.M.S. circuit	alte circuite	C.M.S. circuit	alte circuite						
0	0	0	0	$F/2 \to Y(A)$	Intrare	Intrare	F_3	SIO_3	SIO_3	F_3	F_2	F_1	F_0	Nu	Hi-Z	Hi-Z
0	0	0	1	$F/2 \to Y(L)$	Intrare	Intrare	SIO_3	SIO_3	F_3	F_3	F_2	F_1	F_0	Nu	Hi-Z	Hi-Z
0	0	1	0	$F/2 \to Y(A)$	Intrare	Intrare	F_3	SIO_3	SIO_3	F_3	F_2	F_1	F_0	$Q/2 \to Q(L)$	Intrare	Q_0
0	0	1	1	$F/2 \to Y(L)$	Intrare	Intrare	SIO_3	SIO_3	F_3	F_3	F_2	F_1	F_0	$Q/2 \to Q(L)$	Intrare	Q_0
0	1	0	0	$F \to Y$	Intrare	Intrare	F_3	F_3	F_2	F_2	F_1	F_0	Paritate	Nu	Hi-Z	Hi-Z
0	1	0	1	$F \to Y$	Intrare	Intrare	F_3	F_3	F_2	F_2	F_1	F_0	Paritate	$Q/2 \to Q(L)$	Intrare	Q_0
0	1	1	0	$F \to Y$	Intrare	Intrare	F_3	F_3	F_2	F_2	F_1	F_0	Paritate	$F \to Q$	Hi-Z	Hi-Z
0	1	1	1	$F \to Y$	Intrare	Intrare	F_3	F_3	F_2	F_2	F_1	F_0	Paritate	$F \to Q$	Hi-Z	Hi-Z
1	0	0	0	$2F \to Y(A)$	F_2	F_3	F_3	F_2	F_1	F_1	F_0	SIO_0	Intrare	Nu	Hi-Z	Hi-Z
1	0	0	1	$2F \to Y(L)$	F_3	F_3	F_2	F_2	F_1	F_1	F_0	SIO_0	Intrare	Nu	Hi-Z	Hi-Z
1	0	1	0	$2F \to Y(A)$	F_2	F_3	F_3	F_2	F_1	F_1	F_0	SIO_0	Intrare	$2Q \to Q(L)$	Q_3	Intrare
1	0	1	1	$2F \to Y(L)$	F_3	F_3	F_2	F_2	F_1	F_1	F_0	SIO_0	Intrare	$2Q \to Q(L)$	Q_3	Intrare
1	1	0	0	$F \to Y$	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	Nu	Hi-Z	Hi-Z
1	1	0	1	$F \to Y$	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	$2Q \to Q(L)$	Q_3	Intrare
1	1	1	0	$SIO_0 \to Y_0 - Y_3$	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	Intrare	Nu	Hi-Z	Hi-Z
1	1	1	1	$F \to Y$	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	Nu	Hi-Z	Hi-Z

Notă: A = deplasare aritmetică; L = deplasare logică; Hi-Z = starea a treia; C.M.S. = cel mai semnificativ.

Ieşirea $\overline{WRITE}$ este "0" atunci când se execută o instrucţiune de scriere în RAM. La Am 2903, dacă $\overline{IEN}$ este "1", ieşirea $\overline{WRITE}$ este forţată şi ea pe "1" reţinându-se nemodificat conţinutul registrului Q şi al bistabilului de comparare-semn (figura 1.16). Dacă $\overline{IEN}$ este "0" ieşirea $\overline{WRITE}$ va fi validată, registrul Q şi bistabilul de comparare-semn modificându-se conform instrucţiunii executate. Bistabilul de comparare-semn este un bistabil intern utilizat în timpul operaţiilor de împărţire. Semnalul de comparare-semn apare la ieşirea Z a celui mai semnificativ circuit în timpul funcţiilor speciale C, D, E şi F.

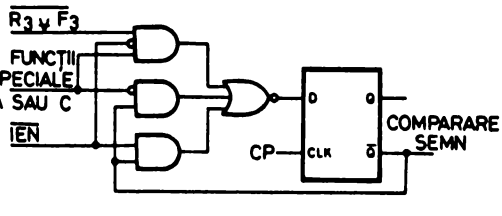


Fig. 1.16. Bistabilul de comparare-semn

La Am 29203, $\overline{IEN}$ controlează scrierea internă fără să afecteze semnalul $\overline{WRITE}$. Semnalul $\overline{IEN}$ poate fi şi în acest fel controlat separat pentru fiecare circuit, facilitându-se astfel operaţiile pe octet.

Prin punerea intrării $\overline{LSS}$ la "0" circuitul se programează pentru a funcţiona în poziţia cea mai puţin semnificativă, validându-se semnalul de ieşire $\overline{WRITE}$ la pinul bidirecţional $\overline{WRITE}/\overline{MSS}$. Dacă intrarea $\overline{LSS}$ este pe "1", pinul $\overline{WRITE}/\overline{MSS}$ devine intrare: prin conectarea lui la "1" circuitul va putea funcţiona într-o poziţie intermediară, iar prin conectarea lui la "0", în poziţia cea mai semnificativă.

1.2.3.3. **Generatorul de transport anticipat Am 2902**

Am 2902 este un generator de transport anticipat de viteză destinat conectării în cascadă a maximum patru UAL binare, acceptând ca intrări perechi de semnale de tip propagare şi generare ($\overline{P}$ şi $\overline{G}$), fig. 1.17. Circuitul are, de asemenea, două ieşiri de tip $\overline{P}$ şi $\overline{G}$ care permit utilizarea lui pentru obţinerea transportului anticipat în structuri cu mai mult de patru microprocesoare *bit-slice*.

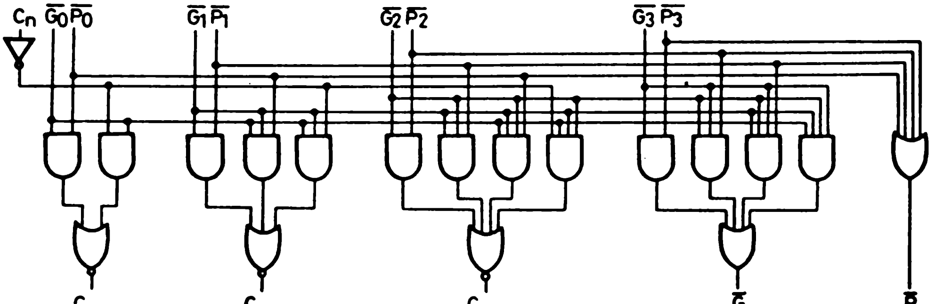


Fig. 1.17. Schema generatorului de transport anticipat Am 2902

Ecuațiile logice ale funcțiilor generate la ieșirile acestui circuit sunt:

$$C_{n+x} = G_0 + P_0C_n;$$
$$C_{n+y} = G_1 + P_1G_0 + P_1P_0C_n;$$
$$C_{n+z} = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_n;$$
$$G = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0;$$
$$P = P_3P_2P_1P_0.$$

1.2.3.4. Circuitul pentru controlul deplasării și al indicatorilor de condiție Am 2904

Am 2904 este un circuit destinat realizării unor operații legate de funcționarea unei UAL, implementate de obicei cu ajutorul unor circuite integrate MSI (fig. 1.18). Printre acestea se numără generarea transportului,

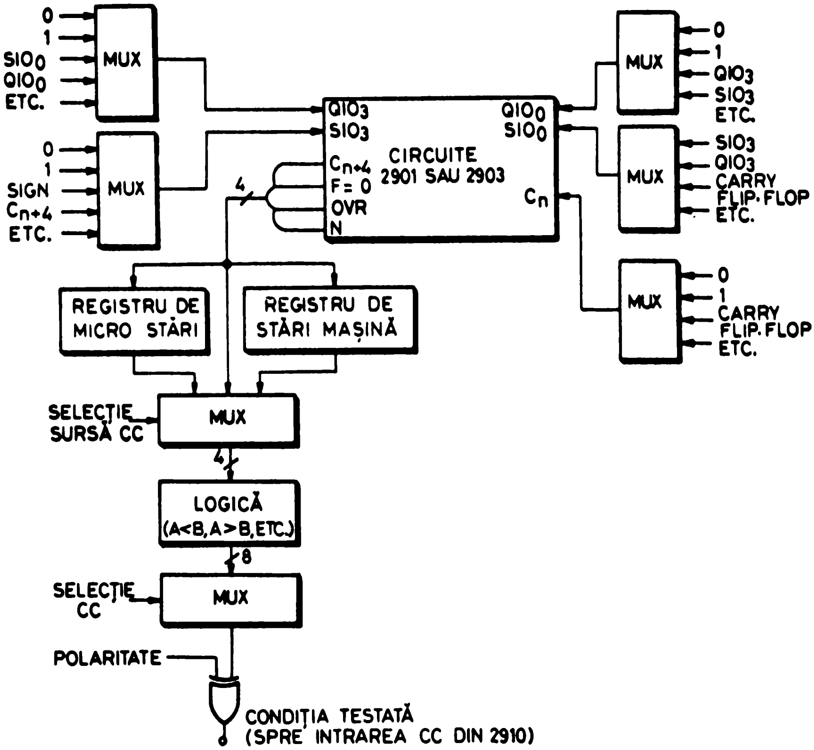


Fig. 1.18. Realizarea cu circuite integrate MSI a funcțiilor lui Am 2904

interconectarea căilor de date, controlul operațiilor de deplasare. Am 2904 realizează aceste funcții cu ajutorul a trei blocuri logice independente. Astfel, transportul se generează cu un multiplexor, iar controlul deplasărilor cu alte patru multiplexoare cu ieșiri de tip "trei-stări". Pentru memorarea indicatorilor de condiție se folosesc două registre. Multiplexorul de testare a condițiilor poate

selecta valoarea adevărată sau negată a indicatorilor de condiţie, precum şi anumite combinaţii ale lor în scopul implementării unor funcţii de test mai complexe.

Schema-bloc a circuitului este dată în figura 1.19. Am 2904 permite implementarea a patru funcţiuni necesare tuturor procesoarelor:

– registrul de stare-program;
– testarea indicatorilor de condiţie;
– controlul deplasărilor;
– controlul transportului.

Cuvântul de comandă al circuitului are 13 biţi notaţi $I_0 - I_{12}$.

Registrul de stare-program. Am 2904 conţine două registre de câte patru biţi în care se pot memora indicatorii de stare ai unei UAL: transportul (C), semnul (N), indicatorul de zero (Z), depăşirea (OVR). Aceste registre sunt numite Registrul de Micro-Stări μSR, *Micro Status Register*, şi Registrul de Stări-Maşină MSR, *Machine Status Register*. Fiecare din aceste registre poate fi controlat independent. Registrele sunt compuse din bistabili D acţionaţi pe frontul pozitiv al ceasului CP.

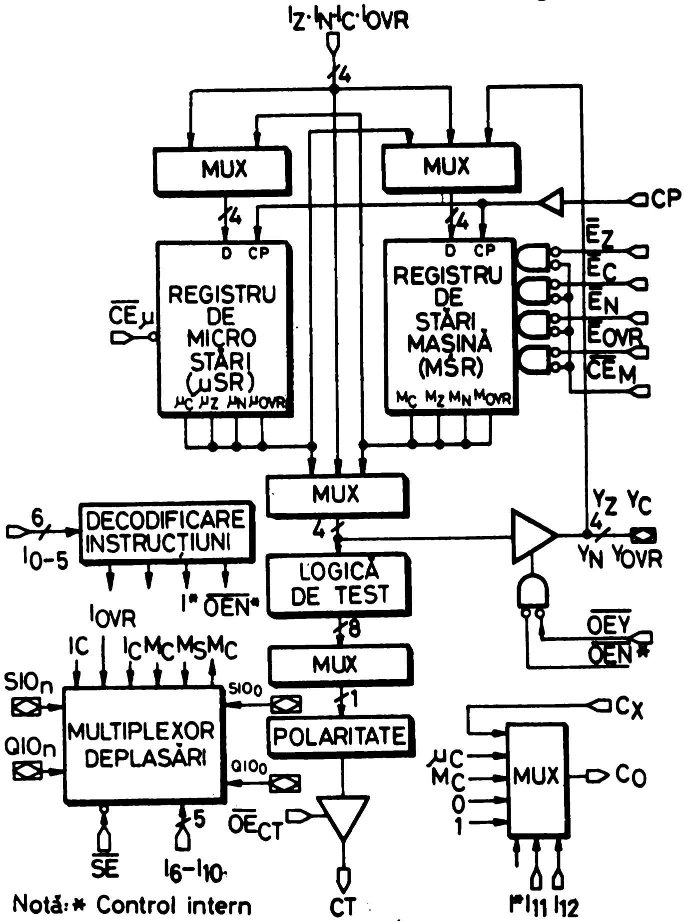


Fig. 1.19. Schema-bloc a circuitului Am 2904

Registrul μSR poate fi încărcat sub controlul biţilor de instrucţiune $I_0 - I_5$ cu indicatorii de condiţie I_C, I_N, I_Z, I_{OVR} sau cu conţinutul registrului MSR. De asemenea, sub controlul biţilor de instrucţiune, biţii registrului μSR pot fi modificaţi individual. Intrarea de validare $\overline{CE}_\mu$ permite inhibarea registrului atunci când este pe "1".

Registrul MSR poate fi încărcat, de asemenea, sub controlul biţilor $I_0 - I_5$, cu indicatorii de condiţie I_C, I_N, I_Z, I_{OVR} cu conţinutul registrului μSR şi cu

valoarea intrărilor/ieşirilor Y_C, Y_N, Y_Z, Y_{OVR}. Registrul MSR mai poate fi şters, poziţionat pe "1" sau complementat. Bistabilii registrului pot fi actualizaţi selectiv controlând cele patru intrări individuale de validare $\overline{E}_C$, $\overline{E}_N$, $\overline{E}_Z$, $\overline{E}_{OVR}$ şi intrarea generală de validare $\overline{CE}_M$. De exemplu, dacă $\overline{CE}_M$ şi $\overline{E}_Z$ sunt "0" se validează actualizarea indicatorului Z. Dacă o intrare de validare este pe "1", atunci se inhibă modificarea indicatorului respectiv. Modificarea întregului registru este inhibată dacă $\overline{CE}_M$ este "1".

Intrările/ieşirile Y_C, Y_N, Y_Z, Y_{OVR} permit citirea registrelor μSR şi MSR în afara circuitului, de exemplu pe magistrala de date a sistemului, precum şi încărcarea registrului MSR din exterior, de exemplu cu informaţia de pe magistrala de date a sistemului. Aceste posibilităţi asigură salvarea şi refacerea registrului stare-program în timpul apelării de subrutine sau în timpul servirii întreruperilor.

Testarea indicatorilor de condiţie. Testarea indicatorilor de condiţie se realizează prin intermediul unui multiplexor care asigură execuţia a 16 funcţii de test diferite. Se pot selecta indicatorii de condiţie nemodificaţi sau negaţi şi combinaţii ale lor, pentru a se executa funcţii de tipul "mai mare", "mai mare sau egal", "mai mic", "mai mic sau egal", pentru numere fără semn sau reprezentate în complement faţă de 2.

Am 2904 poate realiza aceste teste asupra conţinutului registrelor μSR şi MSR sau direct asupra intrărilor I_C, I_N, I_Z, I_{OVR}. Ieşirea de tip "trei-stări" a multiplexorului de testare a condiţiilor, CT, poate fi conectată la intrarea de test-condiţii, $\overline{CC}$, a controlorului de microprogram Am 2910 (§ 1.2.3.7). Ieşirea CT este validată de semnalul $\overline{OE}_{CT}$, ceea ce permite adăugarea altor intrări de test prin cablarea unui SAU LOGIC.

Controlul deplasărilor. Controlul deplasărilor se realizează cu ajutorul unui multiplexor care asigură execuţia a 32 de funcţii de deplasare şi rotaţie diferite. Aceste deplasări şi rotaţii se fac pe lungime simplă sau dublă, cu sau fără transport (M_C). Dacă semnalul $\overline{SE}$ este "1" cele patru intrări/ieşiri SIO_0, SIO_n, QIO_0, QIO_n sunt invalidate. Aceste patru intrări/ieşiri se pot conecta direct la pinii RAM_0, RAM_3, Q_0, Q_3 ai circuitului Am 2901 sau la pinii SIO_0, SIO_3, QIO_0, QIO_3 ai circuitului Am 2903.

Controlul transportului. Am 2904 permite controlul transportului prin intermediul unui multiplexor cu ajutorul căruia pot fi selectate intrările 0, 1, C_x, μ_C, M_C, $\overline{\mu}_C$, $\overline{M}_C$. În acest fel se asigură o implementare mai uşoară a operaţiilor de adunare şi scădere în precizie simplă sau dublă. Intrarea C_x este destinată conectării la ieşirea Z a microprocesorului Am 2903 pentru a se facilita execuţia unora din instrucţiunile speciale ale microprocesorului. Ieşirea C_0 se poate conecta la pinul C_n al celui mai puţin semnificativ Am 2901 sau Am 2903 şi la pinul C_n al generatorului de transport anticipat Am 2902.

Am 2904 este controlat cu ajutorul a 13 biţi de instrucţiune I_0-I_{12} şi a nouă linii de validare $\overline{CE}_M$, $\overline{CE}_\mu$, $\overline{E}_C$, $\overline{E}_N$, $\overline{E}_Z$, $\overline{E}_{OVR}$, $\overline{OE}_Y$, $\overline{OE}_{CT}$, $\overline{SE}$. În majoritatea structurilor microprogramate se urmăreşte o economie de memorie PROM prin micşorarea câmpurilor de control. Acest lucru se poate obţine

pentru Am 2904 prin conectarea la "0" sau "1" a unora din intrările enumerate mai sus, prin conectarea împreună a unora din ele sau prin decodificarea câmpului de microinstrucţiune destinat controlului acestui circuit.

TABELUL 1.10. **Controlul registrului μSR**

Instrucţiuni cu registrul μSR	
$I_5\, I_4\, I_3\, I_2\, I_1\, I_0$ (în octal)	Instrucţiunea
Instrucţiuni la nivel de bit	
1 0	$0 \rightarrow \mu_Z$
1 1	$1 \rightarrow \mu_Z$
1 2	$0 \rightarrow \mu_C$
1 3	$1 \rightarrow \mu_C$
1 4	$0 \rightarrow \mu_N$
1 5	$1 \rightarrow \mu_N$
1 6	$0 \rightarrow \mu_{OVR}$
1 7	$1 \rightarrow \mu_{OVR}$
Instrucţiuni la nivel de registru	
0 0	$M_X \rightarrow \mu_X$
0 1	$1 \rightarrow \mu_X$
0 2	$M_X \rightarrow \mu_X$
0 3	$0 \rightarrow \mu_X$
Instrucţiuni de încărcare	
0 6, 0 7	$I_Z \rightarrow \mu_Z$ $I_C \rightarrow \mu_C$ $I_N \rightarrow \mu_N$ $I_{OVR} + \mu_{OVR} \rightarrow \mu_{OVR}$
3 0, 3 1, 5 0, 5 1. 7 0, 7 1	$I_Z \rightarrow \mu_Z$ $\overline{I}_C \rightarrow \mu_C$ $I_N \rightarrow \mu_N$ $I_{OVR} \rightarrow \mu_{OVR}$
0 4, 0 5, 2 0 _ 2 7, 3 2 _ 4 7, 5 2 _ 6 7, 7 2 _ 7 7	$I_Z \rightarrow \mu_Z$ $I_C \rightarrow \mu_C$ $I_N \rightarrow \mu_N$ $I_{OVR} \rightarrow \mu_{OVR}$

TABELUL 1.11. **Controlul registrului MSR**

Instrucţiuni cu registrul MSR	
$I_5\, I_4\, I_3\, I_2\, I_1\, I_0$ (în octal)	Instrucţiunea
Instrucţiuni la nivel de registru	
0 0	$Y_X \rightarrow M_X$
0 1	$1 \rightarrow M_X$
0 2	$\mu_X \rightarrow M_X$
0 3	$0 \rightarrow M_X$
0 5	$\overline{M}_X \rightarrow M_X$
Instrucţiuni de încărcare	
0 4	$I_Z \rightarrow M_Z$ $M_{OVR} \rightarrow M_C$ $I_N \rightarrow M_N$ $M_C \rightarrow M_{OVR}$
1 0, 1 1, 3 0, 3 1, 5 0, 5 1, 7 0, 7 1,	$I_Z \rightarrow M_Z$ $\overline{I}_C \rightarrow M_C$ $I_N \rightarrow M_N$ $I_{OVR} \rightarrow M_{OVR}$
0 6, 0 7, 1 2 _ 1 7, 2 0 _ 2 7, 3 2 _ 3 7, 4 0 _ 4 7, 5 2 _ 6 7, 7 2 _ 7 7	$I_Z \rightarrow M_Z$ $I_C \rightarrow M_C$ $I_N \rightarrow M_N$ $I_{OVR} \rightarrow M_{OVR}$

TABELUL 1.12. **Controlul ieşirii Y**

$\overline{OE}_Y$	I_5	I_4	Ieşirea Y
1	X	X	Hi-Z
0	0	X	$\mu_i \rightarrow Y_i$
0	1	0	$M_i \rightarrow Y_i$
0	1	1	$I_i \rightarrow Y_i$

Notă: X – fără importanţă; Hi-Z – starea a treia

Biţii $I_0 - I_5$ controlează registrele μSR şi MSR. În tabelul 1.10 se dau codurile operaţiilor executate asupra registrului μSR. Se observă că aceste operaţii sunt împărţite în trei categorii: operaţii asupra biţilor registrului,

operaţii asupra întregului registru şi operaţii de încărcare. Execuţia tuturor acestor operaţii impune ca semnalul de validare $\overline{CE}_{\mu}$ să fie "0".

În tabelul 1.11 se dau codurile operaţiilor executate asupra registrului MSR. Aceste operaţii se împart şi ele în două categorii: operaţii la nivel de registru şi operaţii de încărcare. Execuţia acestor operaţii este condiţionată de poziţionarea semnalului $\overline{CE}_M$ pe "0". Operaţiile la nivel de bit se realizează prin utilizarea operaţiilor la nivel de registru sau a operaţiilor de încărcare şi prin poziţionarea unuia din semnalele de validare $\overline{E}_C$, $\overline{E}_N$, $\overline{E}_Z$, $\overline{E}_{OVR}$.

Biţii I_4-I_5 controlează şi ieşirea Y a circuitului (tabelul 1.12), iar biţii I_0-I_3 ieşirea CT a multiplexorului de testare a indicatorilor de condiţie (tabelul 1.13).

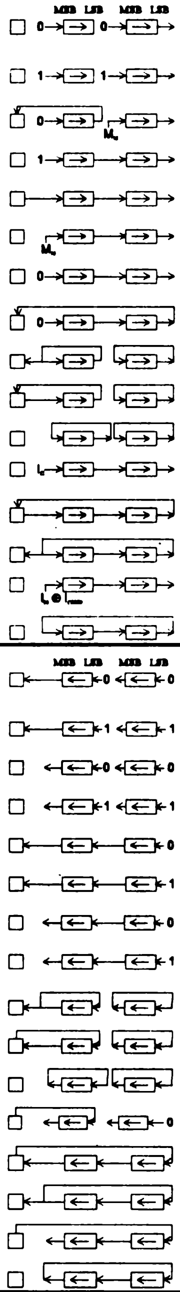
TABELUL 1.13. **Ieşirea CT**

I_3	I_2	I_1	I_0	$I_5=0,\ I_4=0$	$I_5=0,\ I_4=1$	$I_5=1,\ I_4=0$	$I_5=1,\ I_4=1$
0	0	0	0	$(\mu_N \oplus \mu_{OVR})+\mu_Z$	$(\mu_N \oplus \mu_{OVR})+\mu_Z$	$(M_N \oplus M_{OVR})+M_Z$	$(I_N \oplus I_{OVR})+I_Z$
0	0	0	1	$(\mu_N \odot \mu_{OVR})\cdot\overline{\mu}_Z$	$(\mu_N \odot \mu_{OVR})\cdot\overline{\mu}_Z$	$(M_N \odot M_{OVR})+\overline{M}_Z$	$(I_N \odot I_{OVR})+\overline{I}_Z$
0	0	1	0	$\mu_N \oplus \mu_{OVR}$	$\mu_N \oplus \mu_{OVR}$	$M_N \oplus M_{OVR}$	$I_N \oplus I_{OVR}$
0	0	1	1	$\mu_N \odot \mu_{OVR}$	$\mu_N \odot \mu_{OVR}$	$M_N \odot M_{OVR}$	$I_N \odot I_{OVR}$
0	1	0	0	μ_Z	μ_Z	M_Z	I_Z
0	1	0	1	$\overline{\mu}_Z$	$\overline{\mu}_Z$	$\overline{M}_Z$	$\overline{I}_Z$
0	1	1	0	μ_{OVR}	μ_{OVR}	M_{OVR}	I_{OVR}
0	1	1	1	$\overline{\mu}_{OVR}$	$\overline{\mu}_{OVR}$	$\overline{M}_{OVR}$	$\overline{I}_{OVR}$
1	0	0	0	$\mu_C + \mu_Z$	$\mu_C + \mu_Z$	$M_C + M_Z$	$\overline{I}_C + I_Z$
1	0	0	1	$\overline{\mu}_C \cdot \overline{\mu}_Z$	$\overline{\mu}_C \cdot \overline{\mu}_Z$	$\overline{M}_C \cdot \overline{M}_Z$	$I_C \cdot \overline{I}_Z$
1	0	1	0	μ_C	μ_C	M_C	I_C
1	0	1	1	$\overline{\mu}_C$	$\overline{\mu}_C$	$\overline{M}_C$	$\overline{I}_C$
1	1	0	0	$\overline{\mu}_C + \mu_Z$	$\overline{\mu}_C + \mu_Z$	$\overline{M}_C + M_Z$	$\overline{I}_C + I_Z$
1	1	0	1	$\mu_C \cdot \overline{\mu}_Z$	$\mu_C \cdot \overline{\mu}_Z$	$M_C \cdot \overline{M}_Z$	$I_C \cdot \overline{I}_Z$
1	1	1	0	$I_N \oplus M_N$	μ_N	M_N	I_N
1	1	1	1	$I_N \odot M_N$	$\overline{\mu}_N$	$\overline{M}_N$	$\overline{I}_N$

Notă: $\oplus$ SAU EXCLUSIV
$\odot$ SAU NU EXCLUSIV

TABELUL 1.14. **Operaţiile de deplasare**

I_{10}	I_9	I_8	I_7	I_6	M_C RAM Q	SIO_0	SIO_n	QIO_0	QIO_n	Ce se încarcă în M_C
0	0	0	0	0		Hi-Z	0	Hi-Z	0	
0	0	0	0	1		Hi-Z	1	Hi-Z	1	
0	0	0	1	0		Hi-Z	0	Hi-Z	M_N	SIO_0
0	0	0	1	1		Hi-Z	1	Hi-Z	SIO_0	
0	0	1	0	0		Hi-Z	M_C	Hi-Z	SIO_0	
0	0	1	0	1		Hi-Z	M_N	Hi-Z	SIO_0	
0	0	1	1	0		Hi-Z	0	Hi-Z	SIO_0	
0	0	1	1	1		Hi-Z	0	Hi-Z	SIO_0	QIO_0
0	1	0	0	0		Hi-Z	SIO_0	Hi-Z	QIO_0	SIO_0
0	1	0	0	1		Hi-Z	M_C	Hi-Z	QIO_0	SIO_0
0	1	0	1	0		Hi-Z	SIO_0	Hi-Z	QIO_0	
0	1	0	1	1		Hi-Z	I_C	Hi-Z	SIO_0	
0	1	1	0	0		Hi-Z	M_C	Hi-Z	SIO_0	QIO_0
0	1	1	0	1		Hi-Z	QIO_0	Hi-Z	SIO_0	QIO_0
0	1	1	1	0		Hi-Z	$I_N \oplus I_{OVR}$	Hi-Z	SIO_0	
0	1	1	1	1		Hi-Z	QIO_0	Hi-Z	SIO_0	
1	0	0	0	0		0	Hi-Z	0	Hi-Z	SIO_n
1	0	0	0	1		1	Hi-Z	1	Hi-Z	SIO_n
1	0	0	1	0		0	Hi-Z	0	Hi-Z	
1	0	0	1	1		1	Hi-Z	1	Hi-Z	
1	0	1	0	0		QIO_n	Hi-Z	0	Hi-Z	SIO_n
1	0	1	0	1		QIO_n	Hi-Z	1	Hi-Z	SIO_n
1	0	1	1	0		QIO_n	Hi-Z	0	Hi-Z	
1	0	1	1	1		QIO_n	Hi-Z	1	Hi-Z	
1	1	0	0	0		SIO_n	Hi-Z	QIO_n	Hi-Z	SIO_n
1	1	0	0	1		M_C	Hi-Z	QIO_n	Hi-Z	SIO_n
1	1	0	1	0		SIO_n	Hi-Z	QIO_n	Hi-Z	
1	1	0	1	1		M_C	Hi-Z	0	Hi-Z	
1	1	1	0	0		QIO_n	Hi-Z	M_C	Hi-Z	SIO_n
1	1	1	0	1		QIO_n	Hi-Z	SIO_n	Hi-Z	SIO_n
1	1	1	1	0		QIO_n	Hi-Z	M_C	Hi-Z	
1	1	1	1	1		QIO_n	Hi-Z	SIO_n	Hi-Z	



TABELUL 1.15. **Controlul transportului**

I_{12}	I_{11}	I_5	I_3	I_2	I_1	I_0
0	0	X	X	X	X	0
0	1	X	X	X	X	1
1	0	X	X	X	X	C_X
1	1	0	0	X	X	μ_C
1	1	0	X	1	X	μ_C
1	1	0	X	X	1	μ_C
1	1	0	1	0	0	$\overline{\mu}_C$
1	1	1	0	X	X	M_C
1	1	1	X	1	X	M_C
1	1	1	X	X	1	M_C
1	1	1	1	0	0	$\overline{M}_C$

Biţii I_6-I_{10} controlează multiplexorul pentru deplasări (tabelul 1.14), iar biţii $I_{11}-I_{12}$ şi I^* de la decodificatorul intern de instrucţiune, multiplexorul pentru controlul transportului (tabelul 1.15).

1.2.3.5. **Secvenţiatoarele de microprogram Am 2909/2911**

Am 2909/2911 sunt controloare de adresă de 4 biţi destinate controlului adresei de microprogram, deci generării adresei pentru memoria de microprograme. Circuitul poate fi conectat în cascadă pentru a se obţine adrese de 4, 8, 12, 16 ..., biţi.

Schema-bloc a circuitelor este dată în figura 1.20, iar schema detaliată în figura 1.21

Adresa de microprogram este selectată prin intermediul unui multiplexor între intrările directe D, intrările AR memorate într-un registru intern, ieşirea F a unei stive LIFO (*Last In First Out*) de 4 cuvinte şi numărătorul de microprogram μPC. Acest multiplexor de selecţie este controlat de semnalele S_1 şi S_0 (tabelul 1.16a).

TABELUL 1.16a.
Selecţia adresei

S_1	S_0	Informaţia la ieşirile Y_i
0	0	Numărătorul de microprogram (μPC)
0	1	Registrul de adresă (AR)
1	0	Stiva de subrutine (STKO)
1	1	Intrările directe (D)

TABELUL 1.16b.
Operaţiile cu stiva

$\overline{FE}$	PUP	Acţiunea asupra stivei
1	X	Nu se modifică
0	1	Se incrementează *pointer*-ul, se introduce PC în stivă (PUSH)
0	0	Se scoate din stivă (POP)

TABELUL 1.16c.
Ieşirile Y_i

OR_i	$\overline{ZERO}$	$\overline{OE}$	Ieşirile Y_i
X	X	1	Hi-Z
X	0	0	0
1	1	0	1
0	1	0	Selectate de S_1 S_0

Registrul intern de adresă, AR, constă din 4 bistabili de tip D acționați pe frontul pozitiv al ceasului și validați de semnalul $\overline{RE}$. Când acest semnal este "0" în registrul intern se poate înscrie o nouă informație.

Numărătorul de microprogram, μPC, este alcătuit dintr-un registru de 4 biți și un circuit combinațional de incrementare pe 4 biți. Incrementorul are semnale de intrare și ieșire pentru transport (*carry-in* și *carry-out*) care permit conectarea în cascadă a circuitului. Numărătorul de microprogram poate fi utilizat în două moduri. Atunci când cea mai puțin semnificativă intrare de transport din schemă este pe "1", registrul de microprogram este încărcat pe frontul pozitiv al ceasului următor cu adresa curentă incrementată cu 1, (Y + 1). În acest fel microinstrucțiunile se pot executa secvențial una după alta. Dacă cea mai puțin semnificativă intrare de transport este pe "0", incrementorul nu modifică ieșirea Y, astfel încât registrul de microprogram nu-și schimbă conținutul. Așadar, prin utilizarea intrării C_n ca intrare de control o aceeași microinstrucțiune poate fi executată de mai multe ori.

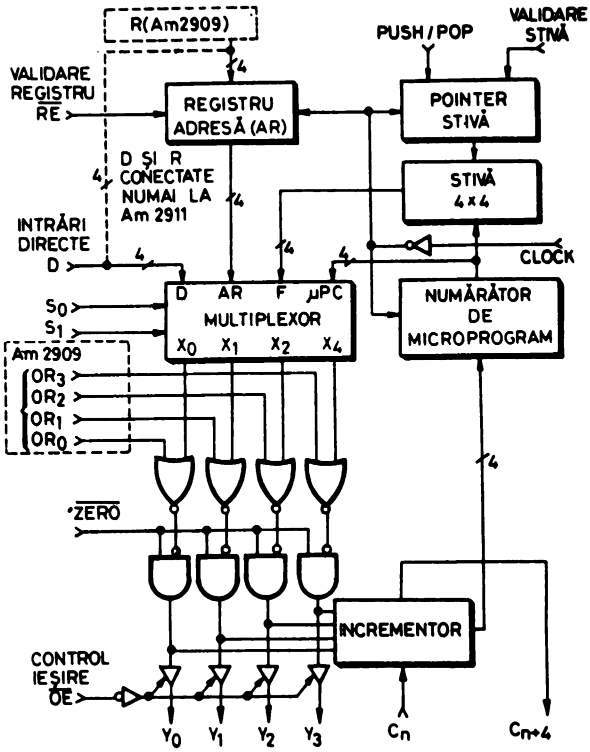
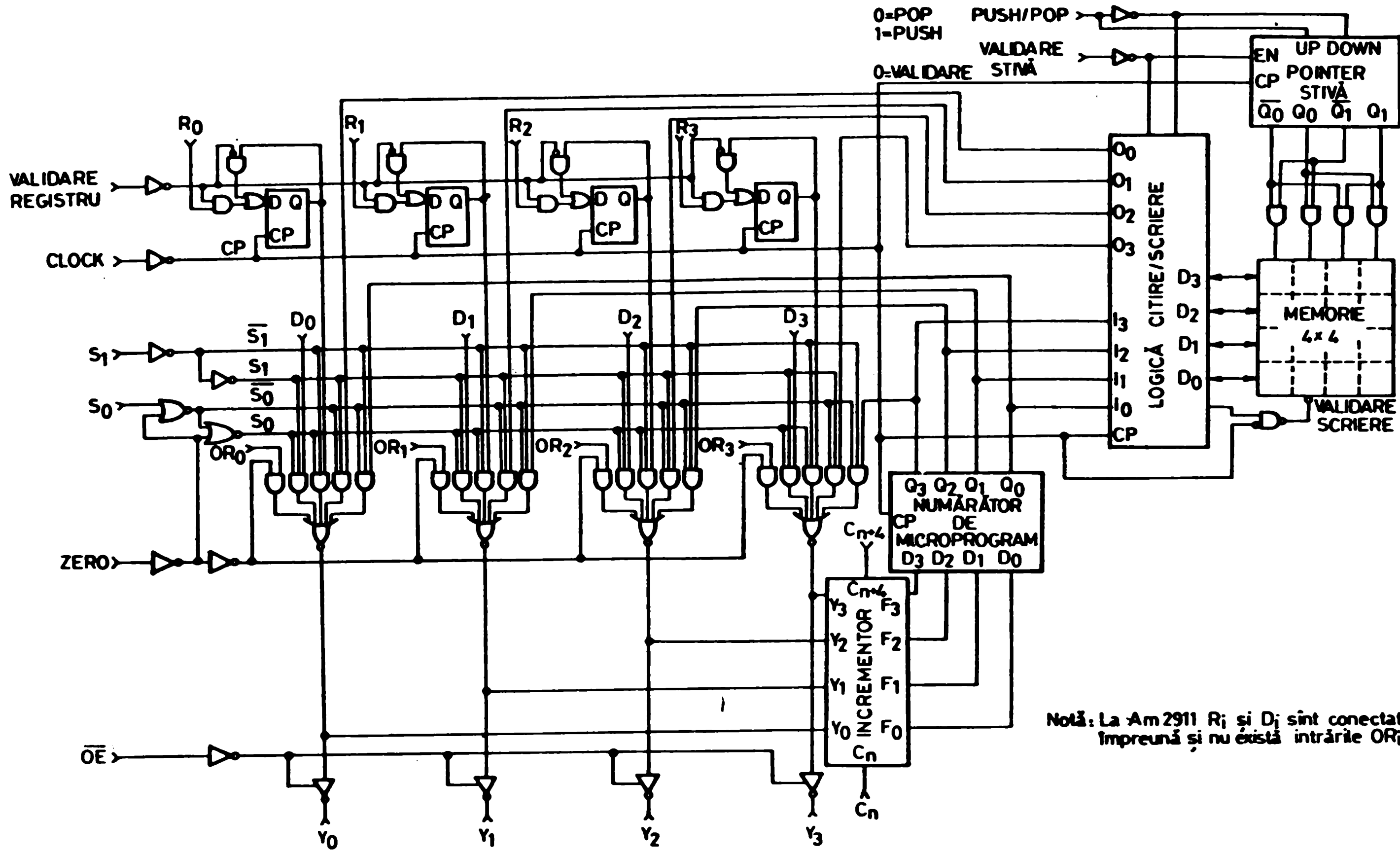


Fig. 1.20. Schema-bloc a secvențiatoarelor Am 2909/2911

Stiva de 4 x 4 biți este folosită pentru memorarea adresei de revenire din microsubrutine. Stiva are atașat un *pointer* (SP) care "punctează" întotdeauna ultima locație scrisă. Acest lucru permite implementarea instrucțiunilor de buclare fără a executa operații de introducere (PUSH) sau scoatere (POP) din stivă (tabelul 1.16b). *Pointer*-ul de stivă funcționează ca un numărător sincron bidirecțional cu intrări separate pentru controlul operațiilor PUSH/POP și al validării stivei, PUP respectiv $\overline{FE}$ (tabelul 1.16b). Dacă intrarea $\overline{FE}$ este "0" și intrarea PUP este "1" se validează o operație de introducere în stivă (PUSH). Aceasta face ca *pointer*-ul de stivă să fie incrementat și adresa de revenire (adresa următoare adresei microinstrucțiunii care a efectuat operația PUSH) să fie scrisă în stivă. Dacă intrările $\overline{FE}$ și PUP sunt "0" se validează o operație de



Notă: La Am 2911 R_i și D_i sînt conectate împreună și nu există intrările OR_i

Fig. 1.21. Schema detaliată a secvențiatoarelor Am 2909/2911

scoatere din stivă (POP). Ieşirea stivei se poate utiliza ca adresă de revenire în ciclul de microinstrucţiune curent. Următorul front pozitiv al ceasului, începutul ciclului următor, va decrementa *pointer*-ul de stivă modificând adresa locaţiei punctate, deci ieşirea stivei. Dacă intrarea de validare $\overline{FE}$ este "1" *pointer*-ul de stivă este inhibat indiferent de valoarea celorlalte intrări, PUP şi CP. Deoarece stiva are o capacitate de patru cuvinte este permisă înlănţuirea (*nesting*) a maximum patru subrutine.

Ieşirea Y a secvenţiatorului, adresa memoriei de microprograme, depinde de semnalele OR, $\overline{ZERO}$ şi $\overline{OE}$ (tabelul 1.16c). Intrarea $\overline{ZERO}$ se utilizează pentru a forţa ieşirile Y la "0" indiferent de valoarea celorlalte intrări, cu excepţia intrării $\overline{OE}$. Intrările OR servesc pentru a forţa ieşirile Y la "1" permiţând implementarea anumitor instrucţiuni de salt. Ieşirile Y ale secvenţiatorului sunt de tip "trei-stări" fiind validate cu ajutorul semnalului $\overline{OE}$. Acest

TABELUL 1.17. **Instrucţiuni de adresare Am 2909/2911**

Ciclu	S_1	S_0	$\overline{FE}$	PUP	μPC	AR	$Stiva_0$	$Stiva_1$	$Stiva_2$	$Stiva_3$	Y	Utilizare
N N+1	0	0 –	0	0	J J+1	K K	R_a R_b	R_b R_c	R_c R_d	R_d R_a	J –	Extragere din stivă, sfârşit de buclă.
N N+1	0	0 –	0	1	J J+1	K K	R_a J	R_b R_a	R_c R_b	R_d R_c	J –	Introducere în stivă, iniţializare de buclă.
N N+1	0	0 –	1	X	J J+1	K K	R_a R_a	R_b R_b	R_c R_c	R_d R_d	J –	*Continue*.
N N+1	0	1 –	0	0	J K+1	K K	R_a R_b	R_b R_c	R_c R_d	R_d R_a	K –	Extragere din stivă şi salt pe AR, sfârşit de buclă.
N N+1	0	1 –	0	1	J K+1	K K	R_a J	R_b R_a	R_c R_b	R_d R_c	K –	Introducere în stivă şi salt pe AR, salt la subrutină (JSR AR).
N N+1	0	1 –	1	X	J K+1	K K	R_a R_a	R_b R_b	R_c R_c	R_d R_d	K –	Salt la adresa din AR, salturi necondiţionate.
N N+1	1	0 –	0	0	J R_a+1	K K	R_a R_b	R_b R_c	R_c R_d	R_d R_a	R_a –	Salt la adresa din stivă şi extragere din stivă, revenire din subrutină.
N N+1	1	0 –	0	1	J R_a+1	K K	R_a J	R_b R_a	R_c R_b	R_d R_c	R_a –	Salt la adresa din stivă şi introducere în stivă.
N N+1	1	0 –	1	X	J R_a+1	K K	R_a R_a	R_b R_b	R_c R_c	R_d R_d	R_a –	Salt la adresa din stivă, revenire din buclă.
N N+1	1	1 –	0	0	J D+1	K K	R_a R_b	R_b R_c	R_c R_d	R_d R_a	D –	Extragere din stivă şi salt la adresa D, sfârşit de buclă.
N N+1	1	1 –	0	1	J D+1	K K	R_a J	R_b R_a	R_c R_b	R_d R_c	D –	Introducere în stivă şi salt la adresa D, salt la subrutină.
N N+1	1	1 –	1	X	J D+1	K K	R_a R_a	R_b R_b	R_c R_c	R_d R_d	D –	Salt la adresa D, salt necondiţionat.

lucru poate fi util dacă se doreşte controlul extern al memoriei de microprograme. Secvenţiatoarele interne se trec în starea a treia, de impedanţă mare, permiţându-se generarea externă a adresei de microprogram cu scopul, de exemplu, de a executa diverse microsubrutine de testare.

În tabelul 1.17 se ilustrează modul în care, cu ajutorul semnalelor S_1, S_0, $\overline{FE}$ şi PUP, se pot obţine diverse instrucţiuni de adresare pentru secvenţiatoarele Am 2909/2911. Cele patru semnale definesc adresa care se generează la ieşirile Y şi starea registrelor interne după aplicarea ceasului, frontul pozitiv. În tabelul 1.17 se presupune că în ciclul N numărătorul de microprogram μPC conţine informaţia J, registrul de adresă AR, informaţia K şi stiva informaţiile $R_a - R_d$. După aplicarea ceasului, deci în ciclul N + 1, starea registrelor interne este definită de cele patru semnale S_1, S_0, $\overline{FE}$, PUP. De exemplu, în cazul instrucţiunii JSR AR, salt la subrutina de la adresa dată de AR, în ciclul N adresa Y este egală cu conţinutul registrului AR, μPC are valoarea J, iar stiva conţine $R_a - R_d$. În ciclul N + 1, μPC va fi egal cu valoarea adresei generate la ieşirile Y incrementată cu 1, deci K + 1, iar stiva va conţine J, $R_a - R_c$, deci adresa de revenire din subrutină.

1.2.3.6. Circuitul de control Am 29811

Circuitul Am 29811 este destinat generării semnalelor de control şi validare pentru secvenţiatoare construite cu Am 2911 [15]. Funcţionarea circuitului este determinată de intrările de instrucţiune $I_3 - I_0$. Circuitul generează

TABELUL 1.18. **Instrucţiuni de adresare Am 2911 realizate cu Am 29811**

Cod mnemonic	I_3	I_2	I_1	I_0	Instrucţiunea
JZ	0	0	0	0	Salt la adresa zero
CJS	0	0	0	1	Salt condiţionat la subrutină cu adresa de salt în registrul *pipeline*
JMAP	0	0	1	0	Salt la adresa de la ieşirea PROM-ului de secvenţe, de *mapare*
CJP	0	0	1	1	Salt condiţionat la adresa din registrul *pipeline*
PUSH	0	1	0	0	Introducere în stivă şi încărcare condiţionată a numărătorului
JSRP	0	1	0	1	Salt la subrutină cu adresa selectată condiţionat între registrul AR din Am 2911 şi registrul *pipeline*
CJV	0	1	1	0	Salt condiţionat la adresa vector
JRP	0	1	1	1	Salt la adresa selectată condiţionat între registrul AR din Am 2911 şi registrul *pipeline*
RFCT	1	0	0	0	Repetă bucla dacă numărătorul este ≠ zero
RPCT	1	0	0	1	Repetă adresa *pipeline* dacă numărătorul ≠ zero
CRTN	1	0	1	0	Revenire condiţionată din subrutină
CJPP	1	0	1	1	Salt condiţionat la adresa din registrul *pipeline* şi extragere din stivă
LDCT	1	1	0	0	Încărcare registru şi *Continue*
LOOP	1	1	0	1	Test sfârşit de buclă
CONT	1	1	1	0	*Continue*
JP	1	1	1	1	Salt la adresa din registrul *pipeline*

semnalele de control S_1, S_0, $\overline{\text{FE}}$, PUP şi semnalele de validare $\overline{\text{COUNTER_LOAD}}$, încărcare numărător, $\overline{\text{COUNTER_ENABLE}}$, validare numărător, $\overline{\text{MAP_ENABLE}}$, validare secvenţă, $\overline{\text{PIPELINE_ENABLE}}$, validare registru *pipeline*. Instrucţiunile de adresare implementate cu Am 29811 sunt date în tabelul 1.18. Aceste instrucţiuni sunt similare cu instrucţiunile de adresare ale secvenţiatorului Am 2910, descrise detaliat în paragraful următor.

1.2.3.7. Controlorul de microprogram Am 2910

Controlorul de microprogram Am 2910 generează o adresă de microprogram de lungime fixă, putând adresa până la 4096 de microinstrucţiuni. Deci, spre deosebire de Am 2909/2911, circuitul nu poate fi conectat în cascadă pentru obţinerea unor adrese de microprogram mai mari de 12 biţi.

Schema-bloc a controlorului de microprogram este dată în figura 1.22.

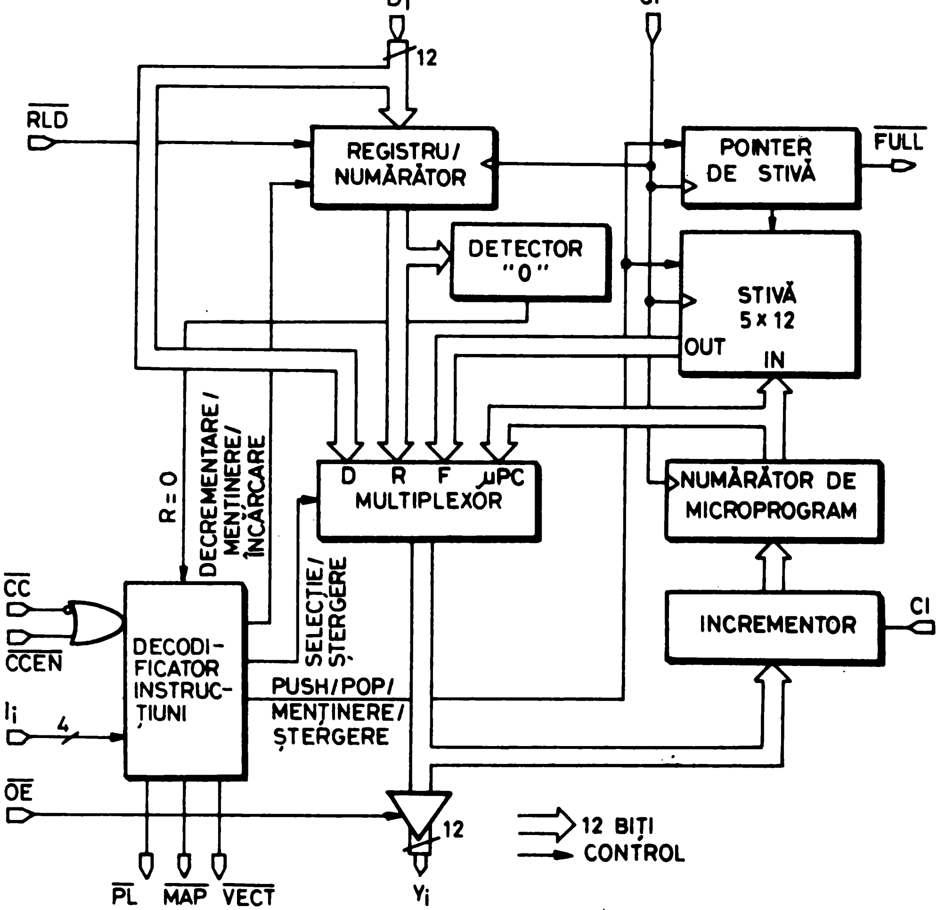


Fig. 1.22. Schema-bloc a controlorului de microprogram Am 2910

Am 2910 implementează 16 instrucţiuni de adresare (tabelul 1.19).

Controlorul este alcătuit dintr-un multiplexor de selectare a adresei, dintr-un registru-numărător, o stivă de 5 cuvinte, un numărător de microprogram şi un PLA pentru decodificarea instrucţiunilor. Multiplexorul poate selecta ca adresă de microprogram intrarea directă D, ieşirea registrului/numărător, ieşirea stivei sau ieşirea numărătorului de microprogram.

TABELUL 1.19. **Instrucţiunile de adresare ale lui Am 2910**

I_3-I_0 (zecimal)	Instrucţiunea	REG./NUM.	Test nereuşit $\overline{CCEN}=0$ şi $\overline{CC}=1$		Test reuşit $\overline{CCEN}=1$ sau $\overline{CC}=0$		REG./NUM.	Validare
			Y	Stiva	Y	Stiva		
0	JZ	X	0	INIT	0	INIT	–	PL
1	CJS	X	PC	–	D	PUSH	–	PL
2	JMAP	X	D	–	D	–	–	MAP
3	CJP	X	PC	–	D	–	–	PL
4	PUSH	X	PC	PUSH	PC	PUSH	Nota 2	PL
5	JSRP	X	R	PUSH	D	PUSH	–	PL
6	CJV	X	PC	–	D	–	–	VECT
7	JRP	X	R	–	D	–	–	PL
8	RFCT	≠0 =0	F PC	– POP	F PC	– POP	DECR –	PL PL
9	RPCT	≠0 =0	D PC	– –	D PC	– –	DECR –	PL PL
10	CRTN	X	PC	–	F	POP	–	PL
11	CJPP	X	PC	–	D	POP	–	PL
12	LDCT	X	PC	–	PC	–	INC	PL
13	LOOP	X	F	–	PC	POP	–	PL
14	CONT	X	PC	–	PC	–	–	PL
15	TWB	≠0 =0	F D	– POP	PC PC	POP POP	DECR –	PL PL

Note: 1. "–" – nu se modifică; X – nu are importanţă; DECR – decrementare; INC – încărcare; INIT – iniţializare.
2. nu se modifică dacă CCEN = 0 şi CC = 1, în celelalte cazuri INC.

Registrul/numărător este alcătuit din 12 bistabili de tip D acționați pe frontul pozitiv al ceasului CP. Dacă semnalul de validare a încărcării $\overline{RLD}$ este pe "0" registrul se încarcă sincron cu informația prezentă la intrările directe D, indiferent de operația specificată de instrucțiunea I_0-I_3.

Registrul/numărător poate funcționa și ca numărător-decrementor, semnalul de zero, generat de detectorul de zero asociat, servind la implementarea instrucțiunilor de buclare. Pentru a executa o secvență de microinstrucțiuni de N ori numărătorul trebuie încărcat cu valoarea $N-1$.

Numărătorul de microprogram este compus dintr-un registru de 12 biți și un incrementor pe 12 biți. Ca și la Am 2909/2911, incrementorul poate fi controlat cu ajutorul intrării de transport CI. Dacă CI este "1" registrul de microprogram este încărcat sincron, pe frontul pozitiv al ceasului, cu adresa curentă incrementată cu 1 $(Y+1)$. Dacă CI este "0" incrementorul nu modifică adresa Y, astfel încât aceeași microinstrucțiune se execută până când CI devine "1".

A treia sursă pentru adresa de microprogram este ieșirea stivei utilizată pentru revenirea din microsubrutine sau pentru execuția buclelor de microprogram. Ca și la Am 2909/2911 stiva este adresată de un *pointer* de stivă care punctează ultima locație scrisă, ceea ce permite buclări de microprogram fără efectuarea unei operații de scoatere din stivă (instrucțiunile RFCT, RPCT). *Pointer*-ul de stivă funcționează ca un numărător bidirecțional incrementat de o operație PUSH și decrementat de o operație POP. Capacitatea stivei, 5 cuvinte, permite apelarea înlănțuită (*nesting*) a maximum 5 subrutine. După apelarea celei de-a cincea subrutine ieșirea $\overline{FULL}$ devine "0" semnalând umplerea stivei. În acest caz orice operație de introducere în stivă (PUSH) scrie numai în vârful stivei, lăsând nemodificat *pointer*-ul de stivă. Această operație, în general distructivă, este de obicei interzisă. Operația de scoatere (POP) din stiva goală este, de asemenea, validă, dar trebuie ținut cont că la ieșirile Y ale controlorului se va plasa o informație fără sens, *pointer*-ul de stivă rămânând pe zero, nemodificat.

Ultima sursă pentru adresa de microprogram este constituită de intrările directe D, utilizate de obicei ca adresă de salt.

Controlorul Am 2910 generează în exterior semnalele de validare $\overline{PL}$, $\overline{MAP}$ si $\overline{VECT}$ cu ajutorul cărora se pot selecta sursele intrării directe D [15]. Semnalul $\overline{PL}$ este destinat pentru selecția primei surse a intrării directe D, de obicei registrul *pipeline* de microinstrucțiune. Semnalul $\overline{MAP}$ selectează a doua sursă, de regulă un PROM sau un PLA pentru generarea adreselor de început, de *mapare*, ale secvențelor de microprogram. Semnalul $\overline{VECT}$ se poate utiliza pentru selecția unei a treia surse a intrărilor D, de exemplu vectorul de întrerupere sau o adresă DMA.

Am 2910 este prevăzut cu o intrare de testare a condițiilor, $\overline{CC}$, la care se poate conecta, de exemplu, ieșirea CT a circuitului Am 2904 (§ 1.2.3.4). Testul este adevărat dacă $\overline{CC}$ este "0". Intrarea $\overline{CCEN}$ invalidează intrarea $\overline{CC}$, controlorul funcționând în starea în care testul este reușit (ca și cum $\overline{CC}$ ar fi "0").

La fel ca la Am 2909/2911, ieşirile Y de tip "trei-stări" permit controlul din exterior al structurii microprogramate a cărei funcţie de secvenţiere este implementată cu Am 2910. $\overline{OE}$ pe "1" invalidează ieşirile controlorului Am 2910 pe busul "trei-stări" de adresă permiţând adresarea din exterior a memoriei de microprograme.

În tabelul 1.19 sunt descrise sintetic instrucţiunile executate de Am 2910 indicându-se starea ieşirilor Y, a semnalelor de validare $\overline{PL}$, $\overline{MAP}$ şi $\overline{VECT}$ generate în exterior, a registrului/numărător şi a stivei. Valoarea efectivă a adresei de microprogram de la ieşirile Y este selectată, aşa cum s-a mai spus, prin intermediul unui multiplexor. Informaţia încărcată în numărătorul de microprogram depinde de valoarea intrării CI, putând fi ieşirea Y sau ieşirea Y incrementată cu 1. Orice instrucţiune validează, poziţionând pe "0", numai una din ieşirile $\overline{PL}$, $\overline{MAP}$ şi $\overline{VECT}$.

Controlorul Am 2910 poate adresa microinstrucţiunea următoare cu ajutorul a 16 instrucţiuni (tabelul 1.19). Aceste instrucţiuni sunt de trei feluri: necondiţionate, al căror efect asupra adresei este dat numai de instrucţiune; condiţionate, al căror efect asupra adresei depinde şi de o informaţie exterioară controlorului; instrucţiuni al căror efect este controlat de conţinutul registrului/ numărător. În cele ce urmează se va presupune că intrarea CI este pe "1", deci că incrementorul funcţionează încărcând Y+1 în registrul de microprogram.

La instrucţiunile de salt condiţionat rezultatul testului de care depinde adresa efectivă se aplică pe intrarea $\overline{CC}$. Dacă această intrare este "0" se consideră că testul este adevărat, reuşit. În acest caz apare acţiunea specificată de numele instrucţiunii. Dacă intrarea $\overline{CC}$ este "1" testul este fals, nereuşit, executându-se o acţiune alternativă, de obicei microinstrucţiunea următoare secvenţial. Testarea internă a intrării $\overline{CC}$ poate fi invalidată prin poziţionarea pe "1", prin microprogram, a semnalului $\overline{CCEN}$. În acest caz se "forţează" acţiunea specificată de numele instrucţiunii. Pentru a nu folosi un bit de microinstrucţiune pentru controlul semnalului $\overline{CCEN}$ se poate recurge la următoarele artificii:

– conectarea la "1" a intrării $\overline{CCEN}$, dacă nu există microinstrucţiuni condiţionate;

– conectarea intrării $\overline{CCEN}$ la "0", dacă instrucţiunile condiţionate nu sunt forţate niciodată să funcţioneze necondiţionat;

– conectarea intrării $\overline{CCEN}$ împreună cu bitul de instrucţiune I_0, ceea ce permite utilizarea instrucţiunilor condiţionate PUSH, CJV şi CRTN.

În instrucţiunile condiţionate, folosite pentru buclări, care utilizează registrul/numărător, acesta este decrementat atât timp cât nu este zero. În ciclul de microinstrucţiune în care registrul/numărător devine zero, instrucţiunea selectează o altă adresă următoare ieşind din bucla de microprogram.

Figura 1.23 ilustrează cele 16 instrucţiuni ale controlorului de microprogram Am 2910, prin indicarea succesiunii adreselor de microprogram. Valoarea adreselor alese nu are nici o importanţă cu excepţia instrucţiunii JZ. Exemplele se referă la o structură de tip *pipeline*, fiecare punct reprezentând timpul cât conţinutul microinstrucţiunii se află în registrul *pipeline*.

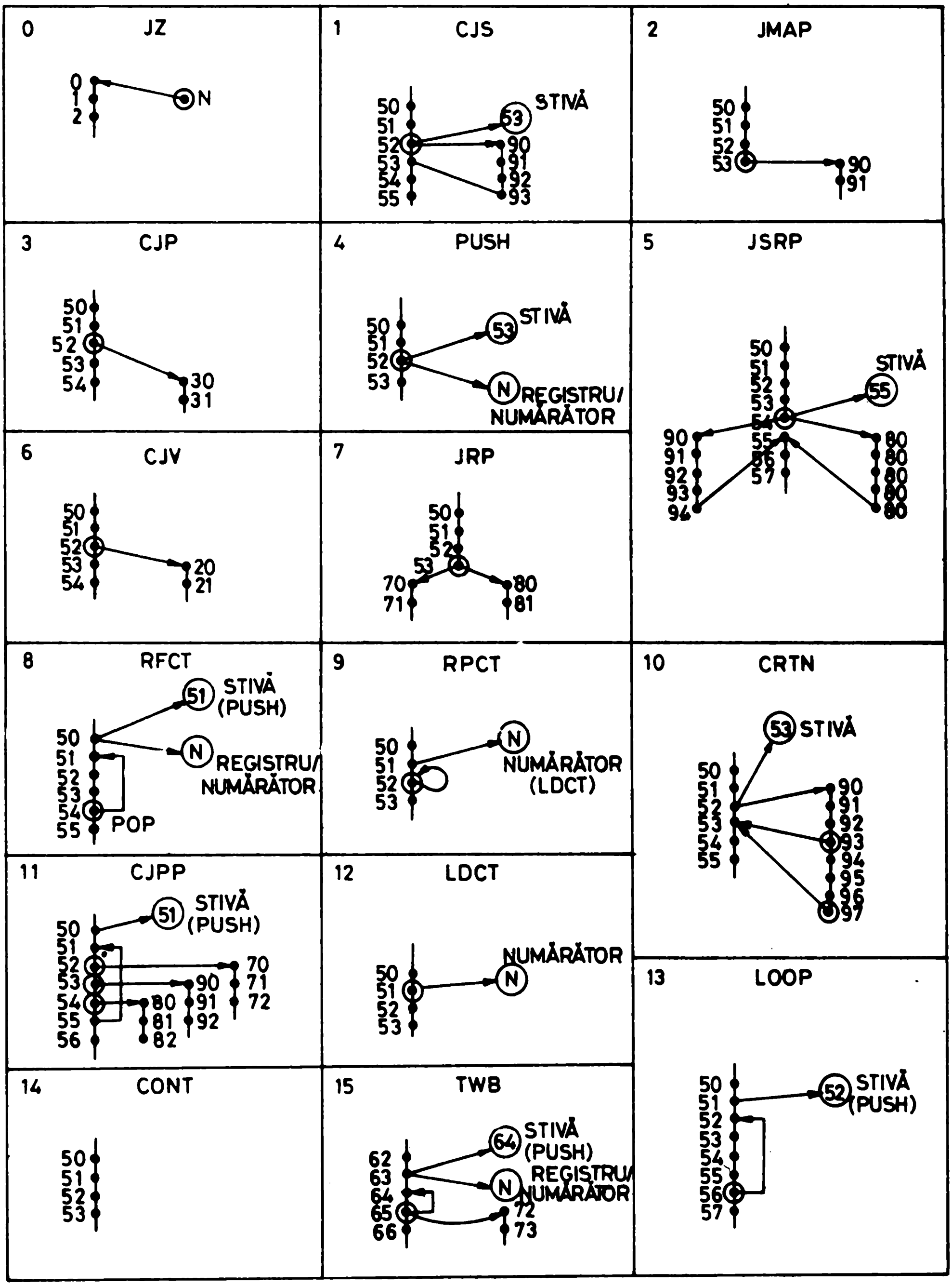


Fig. 1.23. Instrucţiunile de adresare ale controlorului Am 2910

Instrucţiunea 0, JZ (JUMP ZERO sau RESET), este o instrucţiune de salt necondiţionat în care adresa microinstrucţiunii următoare este zero. Această instrucţiune se poate utiliza în secvenţa de punere sub tensiune pentru generarea adresei de început a microprogramului.

Instrucţiunea 1, CJS (CONDITIONAL JUMP-TO-SUBROUTINE), este o instrucţiune de salt condiţionat la adresa din registrul *pipeline*. Când conţinutul microinstrucţiunii de la adresa 52 este în registrul *pipeline* se va executa o instrucţiune CJS. În acest caz adresa următoare depinde de rezultatul unui test. Dacă testul reuşeşte, microinstrucţiunea ce va fi executată în continuare, deci care se va încărca în registrul *pipeline*, va fi cea de la adresa 90 generată de intrările directe D, unde este validat cu ajutorul semnalului $\overline{PL}$ registrul *pipeline* (câmpul de branşare, de constantă sau de adresă următoare al microinstrucţiunii). Adresa 53 a microinstrucţiunii următoare, adresa de revenire din subrutină, este introdusă în stivă. Revenirea din subrutină se face la adresa 93 prin execuţia unei instrucţiuni RETURN-FROM-SUBROUTINE. Dacă testul nu reuşeşte, următoarea microinstrucţiune executată va fi cea de la adresa 53.

Instrucţiunea 2, JMAP (JUMP *MAP*), este o instrucţiune de salt necondiţionat. Adresa de salt este dată de intrările directe D, unde se validează cu ajutorul semnalului $\overline{MAP}$ PROM-ul sau PLA-ul pentru generarea adreselor de început ale diverselor secvenţe de microprogram. Instrucţiunea JMAP se poate utiliza, de exemplu, într-o unitate centrală microprogramată la sfârşitul secvenţei de extragere (FETCH) a instrucţiunii-maşină pentru a continua microprogramul cu secvenţa specifică de tratare a instrucţiunii respective. În exemplul din figura 1.23 secvenţa de extragere se termină cu microinstrucţiunea de la adresa 53 care conţine codul instrucţiunii JMAP. Următoarea microinstrucţiune executată, cea de la adresa 90, va fi microinstrucţiunea de început a unei secvenţe specifice instrucţiunii-maşină citite anterior. Adresa acestei noi microinstrucţiuni este dată la intrările directe D, de un PROM sau un PLA validate de instrucţiunea JMAP.

Instrucţiunea 3, CJP (CONDITIONAL JUMP *P*IPELINE), este o instrucţiune de salt condiţionat. Această instrucţiune permite salturi în microprogram condiţionate de valoarea indicatorilor de condiţie reuniţi la intrarea $\overline{CC}$ a controlorului. După cum se ştie, maşinile microprogramate, maşinile programate în general, testează în diverse puncte de funcţionare anumiţi indicatori, uneori aşteptând ca aceştia să-şi modifice valoarea. Atunci când testul reuşeşte, indicatorul şi-a modificat valoarea iar microprogramul sare pentru a executa altă secvenţă de microinstrucţiuni specifică rezultatului testului. În figura 1.23 saltul condiţionat CJP se execută de microinstrucţiunea de la adresa 52. Adresa următoare va fi dată fie de conţinutul numărătorului de microprogram, fie de intrările directe D la care se validează registrul *pipeline*. Dacă testul reuşeşte adresa următoare este cea din registrul *pipeline* (30). Dacă testul nu reuşeşte adresa următoare este cea dată de numărătorul de program (53).

Instrucţiunea 4, PUSH (*PUSH*/CONDITIONAL LOAD COUNTER), este o instrucţiune utilizată în principal la iniţializarea buclelor de microprogram.

Instrucţiunea PUSH, executată în microinstrucţiunea 52, va introduce adresa următoare (53) în stivă şi va încărca în mod condiţionat registrul/numărător. Dacă testul nu reuşeşte registrul/numărător nu se încarcă. Dacă testul reuşeşte registrul/numărător se va încărca cu valoarea dată la intrările directe D de registrul *pipeline* validat de semnalul $\overline{PL}$. Instrucţiunea RFCT poate utiliza rezultatele instrucţiunii PUSH (adresa introdusă în stivă şi numărul de bucle introdus în registrul/numărător) pentru execuţia efectivă a buclelor de microprogram.

Instrucţiunea 5, JSRP (CONDITIONAL *J*UMP-TO-*S*UBROUTINE *R*EGISTER/*P*IPELINE), este o instrucţiune de salt condiţionat. Instrucţiunea, aşa cum se vede şi în figura 1.23, introduce întotdeauna adresa următoare (55) în stivă şi execută condiţionat una din cele două subrutine (cea de la adresa 80 sau cea de la adresa 90). Pentru ca instrucţiunea JSRP să fie corect executată trebuie avut grijă ca registrul/numărător să fie anterior încărcat cu valoarea dorită. În figura 1.23 registrul/numărător poate fi încărcat, de exemplu, cu valoarea 90, de microinstrucţiunea de la adresa 53. La execuţia microinstrucţiunii de la adresa 54 adresa subrutinei ce va fi executată în continuare depinde de rezultatul testului de condiţie. Dacă testul reuşeşte adresa următoare este dată de câmpul de branşare din registrul *pipeline* (80). Dacă testul nu reuşeşte adresa următoare este dată de conţinutul registrului/numărător (încărcat anterior cu valoarea 90).

Instrucţiunea 6, CJV (CONDITIONAL *J*UMP *V*ECTOR), este o instrucţiune de salt condiţionat la o adresă validată la intrările directe D cu ajutorul semnalului de validare $\overline{VECT}$. Această instrucţiune permite generarea adreselor subrutinelor de tratare a întreruperilor. Pentru că instrucţiunea este condiţionată, adresa următoare depinde de rezultatul testului făcut asupra intrării $\overline{CC}$. Dacă intrarea $\overline{CC}$ este "0", testul reuşit, adresa următoare este generată din exterior la intrările directe D (20). Dacă intrarea $\overline{CC}$ este "1", testul nereuşit, adresa următoare (53) este dată de numărătorul de microprogram.

Instrucţiunea 7, JRP (CONDITIONAL *J*UMP *R*EGISTER/*P*IPELINE), este o instrucţiune de salt condiţionat similară cu instrucţiunea JSR. Diferenţa constă în faptul că instrucţiunea JRP nu execută nici o operaţie asupra stivei. În exemplul din figura 1.23 se presupune că după încărcarea microinstrucţiunii de la adresa 53 în registrul *pipeline*, registrul/numărător este deja încărcat cu valoarea utilă 70. Câmpul de salt al microinstrucţiunii 53 constituie cealaltă adresă utilă (80) execuţiei instrucţiunii JRP. În acest fel, în funcţie de rezultatul testului, microinstrucţiunea executată în continuare va fi cea de la adresa 70, testul nereuşit, sau cea de la adresa 80, testul reuşit.

Instrucţiunea 8, RFCT (*R*EPEAT LOOP, *F*ILE, COUN*T*ER ≠ ZERO), este o instrucţiune de salt condiţionat utilizată în realizarea buclelor de microprogram. Pentru a fi corect executată trebuie ca în prealabil registrul/numărător şi stiva să fie încărcate cu numărul de buclări, respectiv cu adresa de început a buclei. Aceasta se poate face, de exemplu, cu o instrucţiune PUSH. Instrucţiunea RFCT verifică dacă registrul/numărător este diferit de zero. Dacă registrul/numărător este diferit de zero, acesta se decrementează, iar adresa microinstrucţiunii următoare este luată din vîrful stivei (adresa de început a buclei). Operaţia

de citire din stivă nu este însoţită de o decrementare a *pointer*-ului de stivă, ceea ce permite reluarea buclei până când registrul/numărător devine zero. Atunci, când se ajunge la această situaţie, condiţia de ieşire din buclă, adresa microinstrucţiunii următoare este dată de numărătorul de microprogram. Stiva se modifică, se reface, prin execuţia unei operaţii POP. În exemplul din figura 1.23 microinstrucţiunea de la adresa 50 execută o instrucţiune PUSH introducând valoarea N în registrul/numărător (numărul de bucle = N+1) şi adresa 51, adresa de început a buclei, în stivă. Deoarece testarea numărului de bucle se face la sfârşitul buclei în microinstrucţiunea 54, valoarea care trebuie încărcată în registrul/numărător este numărul dorit de buclări plus 1. În acest fel se pot efectua cu ajutorul controlorului Am 2910 bucle de 1 ÷ 4096 ori. Dacă se doreşte să se execute bucle de 0 ÷ 4095 ori, testarea ieşirii din buclă trebuie făcută la începutul buclei.

Instrucţiunea 9, RPCT (*R*EPEAT LOOP, *P*IPELINE REGISTER, *C*OUN*T*ER≠ZERO), este o instrucţiune similară cu instrucţiunea RFCT. Diferenţa constă în faptul că adresa de început a buclei este dată de intrările directe D, unde este validat câmpul de salt din microinstrucţiunea aflată în registrul *pipeline*. Uneori această instrucţiune este folosită ca un mijloc de extensie a stivei deoarece ea permite execuţia unei bucle, chiar după ce stiva este plină. La sfârşitul buclei instrucţiunea RPCT nu modifică stiva. În exemplul din figura 1.23 bucla conţine o singură microinstrucţiune (52) care va executa şi instrucţiunea de adresare RPCT. Câmpul de salt al microinstrucţiunii trebuie să aibă valoarea 52, adresa de început a buclei. Ca şi în cazul instrucţiunii RFCT registrul/numărător trebuie încărcat în prealabil cu numărul dorit de buclări. Acest lucru este făcut de microinstrucţiunea 51 cu ajutorul unei instrucţiuni LDCT.

Instrucţiunea 10, CRTN (*C*ONDITIONAL *RET*UR*N*), este o instrucţiune de salt condiţionat utilizată pentru ieşire din subrutine şi revenire la microinstrucţiunea următoare microinstrucţiunii apelante. Revenirea se face numai dacă testul reuşeşte. Dacă testul nu reuşeşte se execută microinstrucţiunea următoare secvenţial. Deşi este o instrucţiune condiţionată, prin microprogram se poate forţa o funcţionare necondiţionată, poziţionând semnalul $\overline{CCEN}$ pe "1". Apelarea subrutinei în microinstrucţiunea 52 presupune introducerea în stivă a adresei 53 şi continuarea microprogramului cu prima microinstrucţiune a subrutinei apelate (90). Microinstrucţiunea 93 execută o CRTN condiţionată. Dacă testul reuşeşte se efectuează o operaţie POP şi adresa microinstrucţiunii următoare va fi 53. Dacă testul nu reuşeşte, se va executa microinstrucţiunea 94. Microprogramul va continua până la sfârşitul subrutinei unde va întâlni din nou o instrucţiune CRTN. Pentru ca instrucţiunea să fie executată necondiţionat trebuie avut grijă ca microinstrucţiunea respectivă să poziţioneze intrarea de validare $\overline{CCEN}$ pe "1".

Instrucţiunea 11, CJPP (*C*ONDITIONAL *J*UMP *P*IPELINE AND *P*OP), este o instrucţiune de salt condiţionat şi poate fi utilizată la ieşirea din buclele de microprogram sau în diverse operaţii cu stiva. În figura 1.23 bucla de microprogram este formată din microinstrucţiunile 51-55. Microinstrucţiunile

de la adresele 52, 53, 54 sunt toate de tip CJPP. În acest fel se pot asigura mai multe puncte de ieşire condiţionată dintr-o buclă de microprogram. De exemplu, la adresa 52 dacă intrarea $\overline{CC}$ este pe "0", testul reuşit, se va face un salt la adresa 70. *Pointer*-ul de stivă va fi decrementat păstrându-se astfel corectitudinea stivei (anterior, de exemplu, la adresa 50, se introdusese în stivă adresa de început a buclei). Dacă testul nu reuşeşte, $\overline{CC}$ = 1, se va executa microinstrucţiunea 53. În acelaşi fel se poate ieşi din buclă la adresele 53 şi 54. Instrucţiunea CJPP este foarte utilă atunci când bucla de microprogram testează mai mulţi indicatori aşteptând modificarea unuia dintre ei. Instrucţiunea CJPP permite, de asemenea, utilizarea comodă în implementarea microprogramelor a cunoscutei tehnici de programare cu tabele de salturi.

Instrucţiunea 12, LDCT (*LOAD COUNTER* AND CONTINUE), este o instrucţiune necondiţionată care permite încărcarea registrului/numărător cu informaţia prezentă la intrările directe D. Intrările D sunt de obicei conectate la câmpul de constantă al microinstrucţiunii aflate în registrul *pipeline*. Acest câmp de constantă poate servi ca adresă de salt sau pentru încărcarea registrului/numărător. După cum s-a văzut există trei moduri de a încărca registrul/numărător:

– o încărcare simplă cu ajutorul instrucţiunii LDCT;

– o încărcare condiţionată cu instrucţiunea PUSH;

– o încărcare "hardware" cu ajutorul semnalului $\overline{RLD}$ indiferent de instrucţiunea $I_0 - I_3$.

Instrucţiunea LDCT este de fapt o combinaţie între instrucţiunea CONT şi $\overline{RLD}$=0 cu scopul de a asigura o posibilitate simplă de încărcare a registrului/numărător, fără a mai fi nevoie de controlul prin microprogram al intrării $\overline{RLD}$.

Instrucţiunea 13, LOOP (TEST END-OF-*LOOP*), este o instrucţiune de salt condiţionat. LOOP permite ieşirea condiţionată dintr-o buclă, la sfârşitul ei. Dacă testul este reuşit se reia bucla de la adresa dată de ieşirea stivei. Dacă testul nu este reuşit microprogamul se continuă secvenţial. În exemplul din figura 1.23 instrucţiunea LOOP se execută la adresa 56. Dacă testul nu reuşeşte, microprogramul va sări la adresa 52 încărcată anterior în stivă, în microinstrucţiunea de la adresa 51, printr-o instrucţiune PUSH. Dacă testul reuşeşte microprogramul iese din buclă trecând la microinstrucţiunea 57 şi refăcând stiva prin decrementarea *pointer*-ului de stivă.

Instrucţiunea 14, CONT (*CONTINUE*), permite execuţia secvenţială a microprogramului. Această instrucţiune de adresare este cea mai simplă, ea putând fi considerată în unele situaţii şi ca o instrucţiune neoperantă.

Instrucţiunea 15, TWB (*THREE-WAY BRANCH*), constituie o instrucţiune de salt condiţionat. Este cea mai complexă instrucţiune de adresare a controlorului Am 2910 asigurând atât testarea intrării $\overline{CC}$ cât şi a registrului/numărător. Instrucţiunea selectează adresa următoare între ieşirea stivei, registrul *pipeline* şi numărătorul de microprogram. Ca şi în cazul instrucţiunii RFCT trebuie încărcate în prealabil numărul de buclări în registrul/numărător şi adresa de început a buclei în stivă. TWB efectuează ca şi RFCT operaţii de decrementare şi salt până când registrul/numărător devine zero. Adresa de salt este luată din

vârful stivei. După ce registrul/numărător devine zero, adresa următoare este dată de registrul *pipeline*. Acest mod de execuţie corespunde situaţiei în care testul nu reuşeşte. Dacă testul reuşeşte nu se mai face nici un salt şi adresa următoare este dată de numărătorul de microprogram. La ieşirea din buclă, pe registrul/numărător zero sau pe test reuşit, stiva se reface prin decrementarea *pointer*-ului de stivă. În figura 1.23 se dă un exemplu de utilizare a instrucţiunii TWB în cazul unei operaţii de căutare într-un tabel. Microinstrucţiunea 63 execută o operaţie PUSH introducând adresa 64 în stivă şi numărul N în registrul/numărător (lungimea tabelei −1). Microinstrucţiunea de la adresa 64 extrage din tabel elementul următor şi îl compară cu cheia de căutare. Microinstrucţiunea 65 testează rezultatul comparaţiei executând o instrucţiune de adresare TWB. Dacă elementul extras din tabel nu este cel căutat testul nu reuşeşte şi microprogramul revine la adresa 64 pentru o nouă citire în tabel. Dacă registrul/numărător devine zero se sare la adresa 72 unde începe secvenţa specifică situaţiei în care elementul căutat nu a fost găsit în tabel. Dacă elementul căutat se găseşte în tabel microprogramul iese din bucla de căutare executând microinstrucţiunea 66 care reprezintă începutul secvenţei specifice acestei situaţii. La ieşirea din buclă instrucţiunea TWB reface stiva decrementând *pointer*-ul de stivă.

1.2.3.8. Controlorul de program Am 2930

Controlorul de program Am 2930 este un circuit expandabil destinat implementării funcţiei de adresare la nivel de program (adresarea instrucţiunilor) sau la nivel de microprogram (adresarea microinstrucţiunilor).

Schema-bloc a circuitului se dă în figura 1.24.

Circuitul este compus din următoarele elemente principale:

- numărător de program (PC);
- stivă LIFO pentru subrutine (S);
- registrul de lucru (R);
- sumator;
- decodificator de instrucţiuni.

Numărătorul de program este alcătuit dintr-un registru, un incrementor şi un multiplexor de intrare. Registrul de patru bistabili de tip D este acţionat pe frontul pozitiv al ceasului CP. Incrementorul este realizat cu transport anticipat pentru mărirea vitezei circuitului. La conectarea în cascadă ieşirea C_{i+4} a incrementorului se conectează la intrarea de transport C_i a circuitului următor. Ieşirea incrementorului este egală cu intrarea plus C_i. Este deci posibil să se controleze întregul incrementor prin intermediul intrării C_i a celui mai puţin semnificativ circuit. Dacă această intrare este pe "0" incrementorul va lăsa nemodificată ieşirea multiplexorului, iar dacă este pe "1" o va incrementa cu 1. Incrementorul este inhibat intern, în timpul anumitor instrucţiuni de adresare, de semnalul INC. Multiplexorul permite selectarea informaţiei de intrare în incrementor între PC şi sumator.

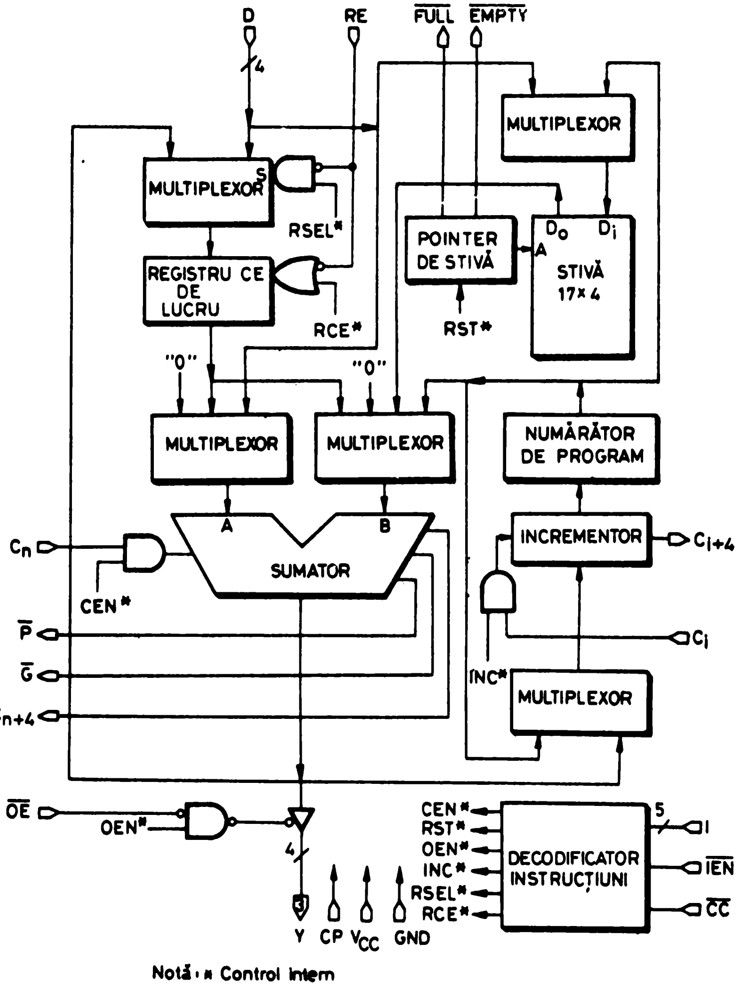


Fig. 1.24. Schema-bloc a controlorului de program Am 2930

Stiva LIFO este alcătuită dintr-un multiplexor de intrare, o memorie RAM de 17 x 4 biți și un *pointer* (SP) pentru adresare. SP punctează întotdeauna ultimul cuvânt scris în RAM, vârful stivei, disponibil deci la ieșirea S a stivei. Informația care se scrie în stivă poate fi selectată cu ajutorul multiplexorului între intrările directe D și PC. Scrierea în stivă se face la adresa SP+1, iar citirea, la adresa SP. *Pointer*-ul SP este un numărător bidirecțional acționat pe frontul pozitiv al ceasului. Atunci când stiva este plină se inhibă incrementarea SP, iar când stiva este goală, decrementarea lui. De asemenea, atunci când stiva este plină se inhibă și operația de scriere în RAM. Când stiva este goală ieșirea $\overline{EMPTY}$ este poziționată pe "0", iar atunci când este plină, ieșirea $\overline{FULL}$ se poziționează pe "0".

Registrul de lucru poate fi încărcat fie cu informaţia prezentă la intrările directe D, fie cu ieşirea sumatorului. Este acţionat pe frontul pozitiv al ceasului şi este validat de intrarea externă $\overline{RE}$ sau, intern, de semnalul RSEL, în timpul instrucţiunilor de adresare. Intrarea $\overline{RE}$ este prioritară validând atunci când este "0", încărcarea în R a intrărilor D, indiferent de instrucţiune.

Sumatorul binar pe patru biţi are, ca şi numărătorul de program, transport anticipat pentru mărirea vitezei circuitului. În vederea obţinerii transportului anticipat într-o conectare în cascadă, Am 2930 generează semnalele $\overline{P}$ şi $\overline{G}$. Pentru sisteme mai lente se poate conecta ieşirea C_{n+4} la intrarea C_n a următorului circuit obţinându-se, în acest fel, o structură cu transport-serie. Multiplexoarele de la intrările sumatorului permit selectarea perechilor de operanzi în funcţie de instrucţiunea de adresare.

Decodificatorul de instrucţiuni generează semnalele necesare stabilirii căilor de informaţie şi validării operaţiilor de încărcare în PC, R, SP, RAM. Intrarea de condiţie $\overline{CC}$ este utilizată pentru decodificare numai în instrucţiuni de salt condiţionat ($\overline{CC}=0$ reprezintă testul reuşit). Intrarea $\overline{IEN}$ de validare-instrucţiune inhibă operaţiile cu PC, SP şi de scriere în stivă, atunci când este pe "1". De asemenea, atunci când $\overline{IEN}=1$ registrul R este controlat numai de intrarea de validare $\overline{RE}$, indiferent de instrucţiunea de adresare.

Am 2930 poate executa 32 de instrucţiuni de adresare specificate de biţii de instrucţiune I_0-I_4 (tabelul 1.20). Instrucţiunile de adresare ale lui Am 2930 pot fi împărţite în:

- instrucţiuni de extragere necondiţionată;
- instrucţiuni de salt condiţionat;
- instrucţiuni de salt condiţionat la subrutină;
- instrucţiuni de revenire condiţionată din subrutină;
- instrucţiuni diverse.

Instrucţiunile de extragere necondiţionată ($1\div9$) permit citirea (extragerea) la ieşirile Y ale circuitului a unor elemente din interiorul lui (PC, R, D, S). În tabelul 1.20 se observă că pentru toate instrucţiunile de extragere PC este incrementat cu valoarea intrării C_i a celui mai puţin semnificativ circuit (într-o structură cu mai multe Am 2930 conectate în cascadă). Pentru instrucţiunile $1\div7$ registrul de lucru R este controlat de intrarea de validare $\overline{RE}$. Pentru instrucţiunile 8 şi 9 registrul R este încărcat cu PC, respectiv R+D. În timpul instrucţiunilor de extragere stiva nu este modificată.

Instrucţiunile de salt condiţionat ($16\div21$) permit selectarea adresei următoare în funcţie de intrarea de test $\overline{CC}$. Dacă testul reuşeşte, $\overline{CC}=0$, adresa următoare este o funcţie de elementele din interiorul controlorului (R, D, PC, "0"). Se observă în tabelul 1.20 că această funcţie, selectată prin instrucţiunea de adresare, eventual incrementată cu 1, este încărcată în PC. Dacă testul nu reuşeşte, $\overline{CC}=1$, se va executa o operaţie FETCH PC, adresa următoare fiind dată de PC. Instrucţiunile de salt condiţionat nu modifică stiva.

Instrucţiunile de salt condiţionat la subrutină ($22\div27$) permit apelarea condiţionată a subrutinelor. Dacă testul nu reuşeşte, $\overline{CC}=1$, adresa următoare

TABELUL 1.20. **Instrucţiunile de adresare ale controlorului Am 2930**

Cod mnemonic	I_4	I_3	I_2	I_1	I_0	$\overline{CC}$	$\overline{IEN}$	Instrucţiunea	Y_{0-3}	Starea următoare: PC	Starea următoare: R, $\overline{RE}=0$	Starea următoare: R, $\overline{RE}=1$	Starea următoare: RAM	Starea următoare: SP
	X	X	X	X	X	X	1	Invalidare	Nota 2	–	D	–	–	–
PRST	0	0	0	0	0	X	0	RESET	"0"	"0"+C_i	D	–	–	Reset
FPC	0	0	0	0	1	X	0	FETCH PC	PC	PC+C_i	D	–	–	–
FR	0	0	0	1	0	X	0	FETCH R	R	PC+C_i	D	–	–	–
FD	0	0	0	1	1	X	0	FETCH D	D	PC+C_i	D	–	–	–
FRD	0	0	1	0	0	X	0	FETCH R+D	R+D+C_n	PC+C_i	D	–	–	–
FPD	0	0	1	0	1	X	0	FETCH PC+D	PC+D+C_n	PC+C_i	D	–	–	–
FPR	0	0	1	1	0	X	0	FETCH PC+R	PC+R+C_n	PC+C_i	D	–	–	–
FSD	0	0	1	1	1	X	0	FETCH S+D	S+D+C_n	PC+C_i	D	–	–	–
FPLR	0	1	0	0	0	X	0	FETCH PC→R	PC	PC+C_i	PC	PC	–	–
FRDR	0	1	0	0	1	X	0	FETCH R+D→R	R+D+C_n	PC+C_i	R+D+C_n	R+D+C_n	–	–
PLDR	0	1	0	1	0	X	0	LOAD R	PC	PC+C_i	D	D	–	–
PSHP	0	1	0	1	1	X	0	PUSH PC	PC	PC+C_i	D	–	PC→(SP+1)	SP+1
PSHD	0	1	1	0	0	X	0	PUSH D	PC	PC+C_i	D	–	D→(SP+1)	SP+1
POPS	0	1	1	0	1	X	0	POP S	S	PC+C_i	D	–	–	SP−1
POPP	0	1	1	1	0	X	0	POP PC	PC	PC+C_i	D	–	–	SP−1
PHLD	0	1	1	1	1	X	0	HOLD	PC	–	D	–	–	–
	1	X	X	X	X	1	0	Test nereuşit	PC	PC+C_i	D	–	–	–

TABELUL 1.20. (*continuare*)

Cod mnemonic	I_4	I_3	I_2	I_1	I_0	$\overline{CC}$	$\overline{IEN}$	Instrucţiunea	Y_{0-3}	Starea următoare				
										PC	R		RAM	SP
											$\overline{RE} = 0$	$\overline{RE} = 1$		
JMPR	1	0	0	0	0	0	0	JUMP R	R	$R+C_i$	D	–	–	–
JMPD	1	0	0	0	1	0	0	JUMP D	D	$D+C_i$	D	–	–	–
JMPZ	1	0	0	1	0	0	0	JUMP "0"	"0"	$"0"+C_i$	D	–	–	–
JPRD	1	0	0	1	1	0	0	JUMP R+D	$R+D+C_n$	$R+D+C_n+C_i$	D	–	–	–
JPPD	1	0	1	0	0	0	0	JUMP PC+D	$PC+D+C_n$	$PC+D+C_n+C_i$	D	–	–	–
JPPR	1	0	1	0	1	0	0	JUMP PC+R	$PC+R+C_n$	$PC+R+C_n+C_i$	D	–	–	–
JSBR	1	0	1	1	0	0	0	JSB R	R	$R+C_i$	D	–	PC→(SP+1)	SP+1
JSBD	1	0	1	1	1	0	0	JSB D	D	$D+C_i$	D	–	PC→(SP+1)	SP+1
JSBZ.	1	1	0	0	0	0	0	JSB "0"	"0"	$"0"+C_i$	D	–	PC→(SP+1)	SP+1
JSRD	1	1	0	0	1	0	0	JSB R+D	$R+D+C_n$	$R+D+C_n+C_i$	D	–	PC→(SP+1)	SP+1
JSPD	1	1	0	1	0	0	0	JSB PC+D	$PC+D+C_n$	$PC+D+C_n+C_i$	D	–	PC→(SP+1)	SP+1
JSPR	1	1	0	1	1	0	0	JSB PC+R	$PC+R+C_n$	$PC+R+C_n+C_i$	D	–	PC→(SP+1)	SP+1
RTS	1	1	1	0	0	0	0	RETURN S	S	$S+C_i$	D	–	–	SP–1
RTSD	1	1	1	0	1	0	0	RETURN S+D	$S+D+C_n$	$S+D+C_n+C_i$	D	–	–	SP–1
CHLD	1	1	1	1	0	0	0	HOLD	PC	–	D	–	–	–
PSUS	1	1	1	1	1	0	0	SUSPEND	Hi-Z	–	D	–	–	–

Nota 1: PC – numărător de program; SP – *pointer*-ul de stivă;
R – registrul auxiliar; D – intrările directe; Hi-Z – starea a treia; "–" – nici o modificare.

Nota 2: Ieşirile *Y* sunt determinate ca şi în cazul $\overline{IEN} = 0$ de $I_0 - I_4$ şi $\overline{CC}$

este dată de PC; stiva nu se modifică. Dacă testul reuşeşte, $\overline{CC}=0$, la ieşirile Y ale circuitului se plasează o funcţie de elementele din interiorul lui (R, D, PC, "0"). Ieşirile Y, eventual incrementate (în funcţie de C_i) sunt încărcate în PC, PC fiind încărcat în stivă la locaţia SP+1 – operaţie prin care se incrementează şi *pointer*-ul. În timpul acestor instrucţiuni registrul R este controlat prin intermediul intrării de validare $\overline{RE}$.

Instrucţiunile de revenire condiţionată din subrutină (28, 29) permit ieşirea condiţionată din subrutină. Dacă testul nu reuşeşte, $\overline{CC}=1$, adresa următoare este dată de PC, stiva rămânând nemodificată. Dacă testul reuşeşte, $\overline{CC}=0$, adresa următoare este S, respectiv S+D. Această adresă, eventual incrementată, în funcţie de C_i, este încărcată în PC, iar *pointer*-ul de stivă decrementat. Registrul R este controlat în timpul acestor instrucţiuni prin intermediul intrării de validare $\overline{RE}$.

Instrucţiunile diverse (0, 10÷15, 30, 31) cuprind:

– instrucţiunea PRST (RESET) care forţează ieşirile Y ale circuitului pe zero, încarcă C_i în PC şi şterge SP;

– instrucţiunea PLDR (LOAD R) care încarcă intrările directe D în registrul R şi, eventual, incrementează PC, în funcţie de C_i;

– instrucţiunea PSHP (PUSH PC) care, ca şi FETCH PC (tabelul 1.20), plasează la ieşirile Y numărătorul de program şi, eventual, incrementează PC; PSHP introduce PC în locaţia SP+1 din stivă şi incrementează *pointer*-ul;

– instrucţiunea PSHD (PUSH D), asemănătoare cu PUSH PC, plasează la ieşirile Y numărătorul de program şi, eventual îl incrementează; deosebirea constă în faptul că în stivă se introduce informaţia prezentă la intrările directe D;

– instrucţiunea POPS (POP S) plasează la ieşirile Y informaţia din vârful stivei, decrementează *pointer*-ul şi, eventual, incrementează PC;

– instrucţiunea POPP (POP PC) plasează la ieşirile Y numărătorul de program, decrementează *pointer*-ul de stivă şi, eventual, incrementează PC;

– instrucţiunea PHLD (HOLD) plasează la ieşirile Y numărătorul de program PC lăsând nemodificate numărătorul de program şi stiva;

– instrucţiunea CHLD (CONDITIONAL HOLD) care, atunci când testul este reuşit, $\overline{CC}=0$, acţionează ca PHLD, iar când testul nu este reuşit, $\overline{CC}=1$, ca FPC;

– instrucţiunea PSUS (SUSPEND) care forţează condiţionat ieşirile Y în starea Hi-Z de impedanţă mare, starea a treia.

BIBLIOGRAFIE

1. WILKES, M.V., *The Best Way to Design an Automatic Calculating Machine*, Report of the Manchester University Computer Inaugural Conference, Electrical Engineering Department of Manchester University, Manchester, Anglia, Iulie 1951, republicat în "Computer Design Development-Principal Papers", editor Earl E. Swartzlander, Jr., Hayden Book Co., Rochelle Park, New Jersey, 1976, menţionat în [2].
2. RAUSHER, T.G.; ADAMS, P.M., *Microprogramming: A Tutorial and Survey of Recent Development*, IEEE Transactions on Computers, 1980, C-29, 1, p. 2-20.

3. SALISBURRY, A.B., *Microprogrammable Computer Architecture*, Elsevier, New-York, 1976.
4. HUSSON, S.S., *Microprogramming. Principles and Practices*, Prentice-Hall, Englewood Cliffs, New Jersey, 1970.
5. ROSIN, R.F., *Contemporary Concepts of Microprogamming and Emulation*, Computer Surveys, Vol. 1, Decembrie 1969, menţionat în [2].
6. DAVIDSON, S.; SHRIVER, B.D., *An Overview of Firmware Engineering*, Computer, 1978, 11, 5, p. 21-33.
7. DANCEA, I., *Microprocesoare, arhitectură internă, programare, aplicaţii*, Editura Dacia, Cluj-Napoca, 1979.
8. PETRESCU, A., *Microprogramare. Principii şi aplicaţii*, Editura Tehnică, Bucureşti, 1975.
9. KATZAN, Jr., H., *Microprogramming Primer*, McGraw-Hill Book Company, New York, 1977, p. 103-118, 197-223.
10. STOCKENBERG, J.; DAM, ANDRIES VAN, *Vertical Migration for Performance Enhancement in Layered Hardware/Firmware/Software Systems*, Computer, 1978, 11, 5, p. 35-50.
11. BAER , J.L.; KOYAMA, B., *On the Minimization of the Width of the Control Memory of Microprogrammed Processors*, IEEE Transactions on Computers, 1979, C-28, 4, p. 310-316.
12. LUCIDO, T.P.; CHATTERGY, R; POOCH, U.W., *A survey of Microprogram Verification and Validation Methods*, The Computer Journal 1981, 24, 2, p. 139-142.
13. ROBERTSON, E.L., *Microcode Bit Optimization is NP-Complete*. IEEE Transactions on Computers, 1979, C-28, 4, p. 316-319.
14. ŢĂPUŞ, N., *Contribuţii la elaborarea de noi structuri de sisteme de calcul microprogramate*, Teză de doctorat, Institutul Politehnic Bucureşti, Facultatea Automatică, Bucureşti, 1981.
15. LUPU, C.; ŢEPELEA, V.; PURICE, E., *Microprocesoare. Aplicaţii*, Editura Militară, Bucureşti, 1982.
16. x x x, *The Am 2900 Family Data Book with Related Support Circuits*, Advanced Micro Devices Inc., Sunnyvale, California, 1979.
17. x x x, *Series 3000 Reference Manual*, Santa Clara, California, Intel Corp., 1976.
18. x x x, *Introducing the Series 3000 Bipolar Microprocessor* - Signetics Corp., Sunnyvale, California.
19. TANENBAUM, A.S., *Structured Computer Architecture*, Prentice-Hall, Inc., 1976.
20. HABIB, S., *Microprogramming and firmware engineering methods*, Van Nostrand Reinhold Ltd., 1988.
21. SEGEE, B.E.; FIELD, J., *Microprogramming and Computer Architecture*, John Wiley & Sons Ltd, 1991.
22. HILL, F.J.; PETERSON, G.R., *Digital Logic and Microprocessors*, John Wiley & Sons Ltd, 1984.
23. HILL, F.J.; PETERSON, G.R., *Digital Systems: Hardware Organization and Design*, 3rd ed., John Wiley & Sons Ltd, 1987.
24. BLANCHET, G.; DUPOUY, B., *Architecture des ordinateurs, des techniques de base aux techniques avancées*, Masson, 1991.
25. BROWN, F., *Processeurs Risc. L'exemple de l'Am 29000*, Masson, 1991.
26. LAM, H.; O'MALLEY, J.R., *Fundamentals of Computer Engineering: Logic Design and Microprocessors*, John Wiley & Sons Ltd, 1988.
27. HENNESSY, J.L.; PATTERSON, D.A., *Computer Arhitecture. A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

2

FAMILIA MICROPROCESORULUI Z80

Microprocesorul Z80 este cel mai reprezentativ microprocesor pe 8 biţi. Familia de circuite Z80 conţine unitatea centrală de tipul UC-Z80 şi diverse dispozitive de intrare/ieşire, I/E, de uz general. Funcţiile clasice ale unui sistem cu microprocesor (I/E paralelă, I/E serie, numărare/temporizare şi acces direct la memorie) se pot implementa uşor cu circuite din această familie, conectând la UC-Z80 circuitele PIO-Z80, pentru comanda I/E paralele, SIO-Z80 sau DART-Z80, pentru comanda I/E serie, CTC-Z80, pentru realizarea funcţiilor de numărare/temporizare, şi DMA-Z80 pentru comanda accesului direct la memorie.

Vom prezenta în continuare componentele mai des utilizate ale familiei Z80: UC-Z80, PIO-Z80, CTC-Z80 şi SIO-Z80.

2.1. UNITATEA CENTRALĂ PE 8 BIŢI UC-Z80

În figura 2.1 este reprezentată schema-bloc a unui microprocesor Z80. Procesorul este organizat în jurul unei magistrale interne având ca elemente de bază o unitate aritmetică-logică pe 8 biţi, UAL, registre, circuitele de comandă a magistralelor de date şi adrese, registrul de instrucţiuni împreună cu circuitele pentru decodificarea şi comanda unităţii centrale. Prelucrarea fiecărei instrucţiuni începe cu extragerea ei din memorie şi încărcarea în registrul de instrucţiuni. După decodificare, circuitele de comandă generează toate elementele necesare pentru citirea/scrierea datelor din/în registre, comandă

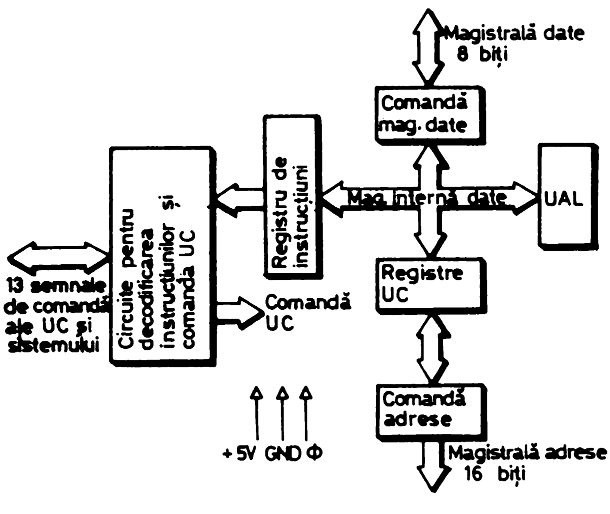


Fig. 2.1. Schema-bloc a unităţii centrale UC-Z80

UAL şi asigură toate semnalele de comandă externe microprocesorului. Registrele interne, accesibile programatorului, sunt împărţite în două seturi de câte şase registre generale ce pot fi folosite individual, ca registre de 8 biţi, sau perechi, ca registre de 16 biţi. Acumulatorul şi registrul cu indicatorii de condiţii sunt, de asemenea, dublate.

UC-Z80 mai conţine un indicator al vârfului de stivă, *Stack Pointer*, un numărător de program, două registre index, un registru numărător pentru reîmprospătarea memoriei dinamice externe şi un registru pentru memorarea întreruperilor. Unitatea centrală are nevoie pentru alimentare de o singură tensiune, +5V. Semnalele externe sunt decodificate şi sincronizate, ceea ce permite interfaţarea directă a microprocesorului cu memorii şi circuite periferice standard.

2.1.1. REGISTRELE UNITĂŢII CENTRALE UC-Z80

În figura 2.2 sunt date cele trei grupuri de registre din cadrul UC-Z80. Primul grup este alcătuit din două seturi identice de registre de 8 biţi, registrele principale (de exemplu A, B) şi registrele secundare (de exemplu A', B'). După cum se vede în figură, ambele seturi sunt compuse dintr-un acumulator, un

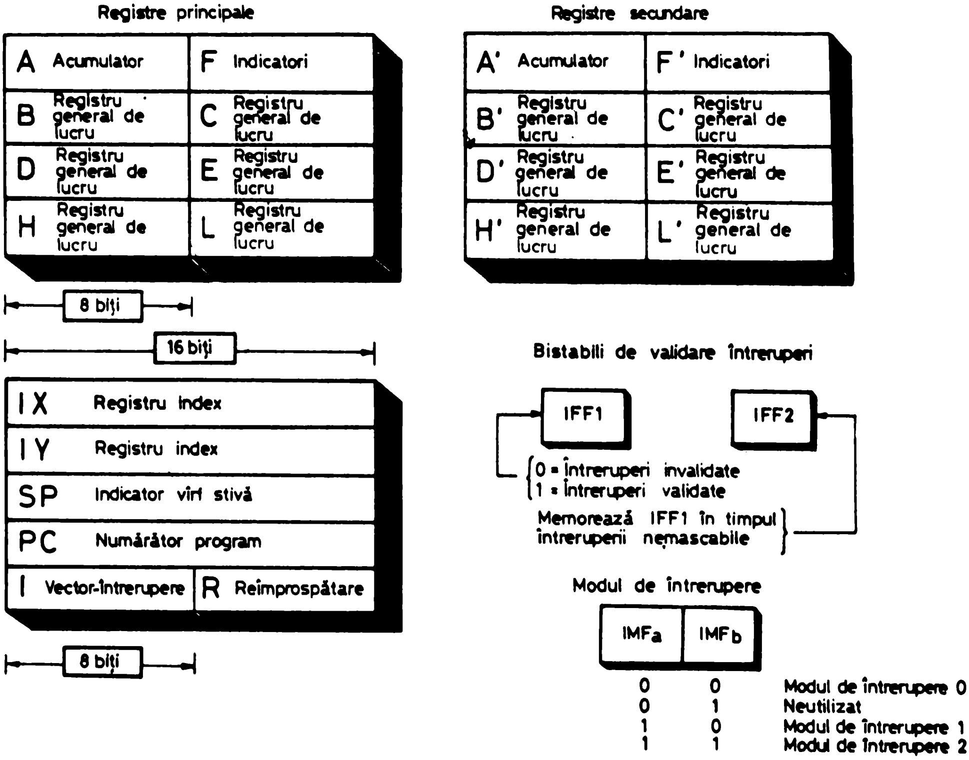


Fig. 2.2. Registrele UC-Z80

registru cu indicatorii de condiţii şi şase registre generale de lucru. Transferul datelor între cele două seturi de registre se poate face cu ajutorul unor instrucţiuni *Exchange*. Existenţa celor două seturi identice de registre permite un răspuns mai rapid la întreruperi şi o implementare eficientă a unor tehnici de programare, cum este prelucrarea în *foreground/background*. De asemenea, se poate afirma că această structură de registre simplifică programarea, în special pentru sistemele cu multă memorie ROM şi puţină memorie RAM.

Al doilea grup de registre constă din şase registre cu funcţii specializate, specifice unităţilor centrale. Aceste registre sunt:

Numărătorul de program, PC, Program Counter; păstrează adresa de 16 biţi a instrucţiunii curente ce se extrage din memorie. PC este incrementat automat după ce conţinutul său se transferă în *buffer*-ul care comandă liniile de adrese externe. Atunci când în program apare un salt, se invalidează acţiunea circuitului intern de incrementare, în PC încărcându-se noua valoare a adresei de salt.

Indicatorul vârfului de stivă, SP, Stack Pointer; păstrează adresa de 16 biţi a vârfului stivei plasate oriunde în memoria RAM externă. Zona din memoria RAM utilizată ca stivă este organizată pe principiul "ultimul-intrat, primul-ieşit" (LIFO – *Last In First Out*).

Registrele index, IX şi IY; cele două registre sunt destinate păstrării unor adrese de bază utilizabile în modul de adresare indexată. În această adresare, registrul index conţine adresa unei *zone* de memorie, baza. Deplasamentul, un număr cu semn exprimat în complement faţă de 2, este specificat prin intermediul unui octet suplimentar adăugat la instrucţiunile indexate.

Registrul pentru vectorul-întrerupere, I; UC-Z80 poate funcţiona într-un mod de tratare al întreruperilor când, ca răspuns la o cerere de întrerupere, se execută o instrucţiune de apel indirect la orice locaţie de memorie. Registrul I păstrează cei mai semnificativi 8 biţi ai adresei indirecte, în timp ce octetul cel mai puţin semnificativ este generat de dispozitivul care întrerupe. Acest mod de tratare a întreruperilor permite plasarea dinamică a subrutinelor de serviciu, oriunde în spaţiul de memorie, timpul de acces rămânând în orice situaţie minim.

Registrul pentru reîmprospătare, R; unul din numeroasele atuuri ale microprocesorului Z80 în faţa celorlalte microprocesoare pe 8 biţi îl constituie posibilitatea conectării directe a memoriilor dinamice. Acest lucru este posibil datorită mecanismului de reîmprospătare implementat în structura UC-Z80. Registrul R de 7 biţi se incrementează după fiecare extragere de instrucţiune. Conţinutul lui se plasează pe porţiunea mai puţin semnificativă a magistralei de adrese, acţiune însoţită de generarea unui semnal de comandă a reîmprospătării. Reîmprospătarea se execută transparent din punct de vedere al utilizatorului, pe timpul cât unitatea centrală decodifică şi execută instrucţiunea extrasă, când magistrala este liberă.

Bistabilii de validare întreruperi, IFF1 şi IFF2; IFF1 semnalează starea de validare sau invalidare a întreruperilor în timp ce IFF2 stochează temporar valoarea lui IFF1, pe timpul tratării întreruperii nemascabile. Poziţionarea celor

doi bistabili pe "1" şi pe "0" se face prin program cu ajutorul instrucţiunilor EI, respectiv DI.

Bistabilii de mod întrerupere, IMFa şi IMFb; codifică unul din cele trei moduri de lucru în întreruperi ale UC-Z80. Stabilirea modului de lucru se face prin program cu ajutorul instrucţiunilor IM 0, IM 1 sau IM 2.

2.1.2. SISTEMUL DE ÎNTRERUPERI

UC-Z80 acceptă două semnale de întrerupere: $\overline{NMI}$, întreruperea nemascabilă, şi $\overline{INT}$, întrerupere mascabilă validată selectiv prin program. La întreruperea nemascabilă Z80 răspunde într-un singur mod, în timp ce pentru întreruperea mascabilă există trei moduri de tratare. $\overline{NMI}$ este prioritară faţă de $\overline{INT}$.

La iniţializare bistabilii IFF1 şi IFF2 sunt forţaţi pe zero, ceea ce echivalează cu invalidarea întreruperilor; în această stare microprocesorul nu acceptă întreruperi mascabile. Întreruperile se validează prin poziţionarea bistabililor IFF1 şi IFF2 pe "1" cu ajutorul instrucţiunii EI. Orice întrerupere în aşteptare va putea fi servită numai după execuţia instrucţiunii care urmează după EI. Întârzierea de o instrucţiune este utilă atunci când după EI se execută o instrucţiune de revenire. În cazul în care UC-Z80 acceptă o întrerupere, IFF1 şi IFF2 sunt aduşi pe "0", inhibându-se astfel acceptarea unor alte întreruperi până la o nouă instrucţiune EI.

Aşa cum am menţionat, destinaţia lui IFF2 este de a memora temporar starea lui IFF1 la apariţia unei întreruperi nemascabile când, pentru prevenirea celorlalte întreruperi, IFF1 se forţează pe "0". Mai mult, la execuţia unei instrucţiuni LD A, I sau LD A, R starea lui IFF2 este transcrisă în indicatorul de paritate, ceea ce permite testarea sau memorarea ei şi deci, implicit, refacerea prin program a valorii iniţiale a lui IFF1. O altă cale, cea obişnuită, de a reface starea precedentă întreruperii nemascabile este prin execuţia unei instrucţiuni de revenire din întreruperea nemascabilă, RETN.

Întreruperea nemascabilă nu poate fi invalidată prin program fiind acceptată în orice situaţie de UC-Z80. $\overline{NMI}$ se rezervă de obicei pentru evenimente prioritare, cum este căderea de tensiune. La apariţia semnalului $\overline{NMI}$, dacă semnalul $\overline{BUSRQ}$ nu este activ, microprocesorul ignoră în ciclul de extragere următor codul instrucţiunii iniţiind un restart la locaţia 0066H. La această adresă se găseşte, în general, secvenţa de serviciu a întreruperii nemascabile.

Z80 poate fi programat pentru a răspunde la întreruperile mascabile într-unul din modurile 0, 1 sau 2, memorate cu ajutorul bistabililor IMFa şi IMFb. Cele trei moduri sunt descrise în continuare.

Modul 0. Este compatibil cu procedurile de întrerupere ale microprocesorului 8080. În acest mod de întrerupere dispozitivul periferic poate plasa pe magistrala de date, în ciclul de tratare a întreruperii, orice instrucţiune. Deci, ideea procedurală este că instrucţiunea următoare nu se mai extrage din

memorie, fiind furnizată de dispozitivul care întrerupe. În general, aceasta este o instrucţiune *restart*, deoarece perifericul trebuie să asigure plasarea pe magistrala de date a unui singur octet. Instrucţiunile de restart realizează apeluri de subrutine plasabile la opt locaţii fixe în pagina zero de memorie. Desigur, dispozitivul care întrerupe poate genera orice cod de instrucţiune, de exemplu o instrucţiune CALL, formată din trei octeţi pentru apel la orice locaţie de memorie. Precizăm, de asemenea, că la iniţializare UC-Z80 intră automat în modul 0.

Modul 1. Este foarte asemănător cu modul de răspuns la întreruperea nemascabilă. Diferenţa principală constă în faptul că se execută un restart la locaţia 0038H în loc de 0066H.

Modul 2. Este cel mai puternic mod de răspuns al microprocesorului Z80: cu un singur octet furnizat de dispozitivul care întrerupe se poate executa un *apel indirect* la orice locaţie de memorie.

Pentru a folosi acest mod de tratare a întreruperilor programatorul trebuie să scrie o tabelă cu adresele de început ale fiecărei rutine de serviciu. Tabela poate fi localizată în orice zonă a memoriei. La acceptarea unei întreruperi UC-Z80 formează un *pointer* de 16 biţi cu ajutorul căruia ia din tabelă adresa rutinei de serviciu corespunzătoare dispozitivului care întrerupe. Cei mai semnificativi 8 biţi ai *pointer*-ului sunt daţi de conţinutul registrului I încărcat în prealabil (la iniţializare I=0). Cei mai puţin semnificativi 8 biţi vor fi generaţi de periferic, cu observaţia că, deoarece ultimul bit trebuie să fie zero, sunt necesari de fapt numai 7. De aici rezultă că adresele de început ale subrutinelor de serviciu vor fi plasate în tabelă întotdeauna la adrese pare. În figura 2.3 se înfăţişează procedura de tratare a întreruperii în modul 2: după ce dispozitivul periferic care întrerupe generează porţiunea cea mai puţin semnificativă a *pointer*-ului, UC-Z80 salvează automat în stivă numărătorul de program, obţine din tabelă adresa de început a subrutinei de serviciu şi efectuează un salt la această adresă.

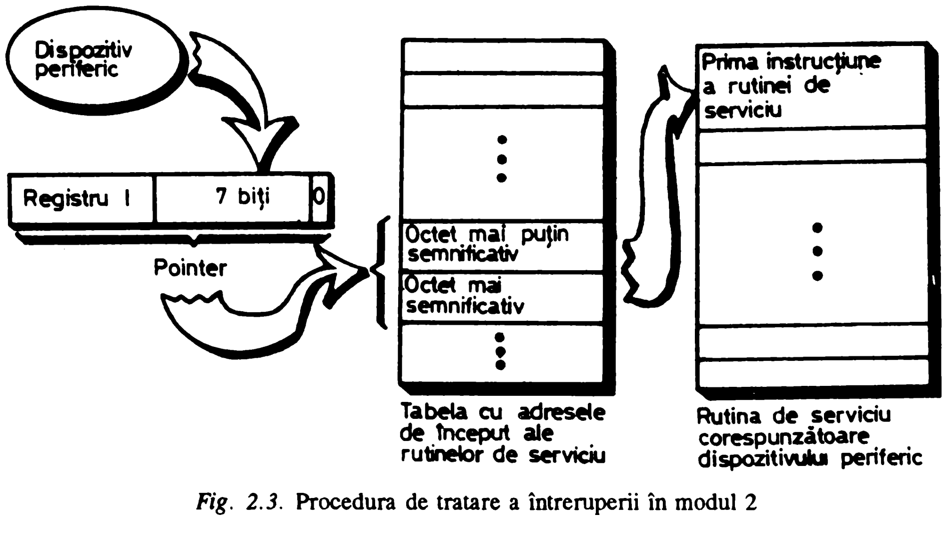


Fig. 2.3. Procedura de tratare a întreruperii în modul 2

Dispozitivele periferice din seria Z80 permit implementarea unui sistem de întreruperi care să lucreze în modul 2 asigurând în acest scop generarea automată a vectorului de întrerupere pe timpul unui ciclu de achitare. De asemenea, menţionăm că dispozitivele care întrerup pot fi conectate în lanţ, prioritatea fiind determinată de poziţia fizică în lanţ. Fiecare circuit are o intrare de validare a întreruperilor, IEI, şi o ieşire de validare a întreruperilor, IEO, către următorul dispozitiv. Primul dispozitiv din lanţ, prioritar, trebuie să aibă intrarea IEI cablată la "1". Comanda semnalului IEO în funcţie de IEI şi de întreruperea locală este asigurată de toate circuitele periferice din seria Z80.

2.1.3. DESCRIEREA CONEXIUNILOR EXTERNE ALE UC-Z80

UC-Z80 este fabricat într-o capsulă de tip DIL, *Dual In-Line*, cu 40 de conexiuni externe. În figura 2.4 sunt indicate toate aceste conexiuni externe

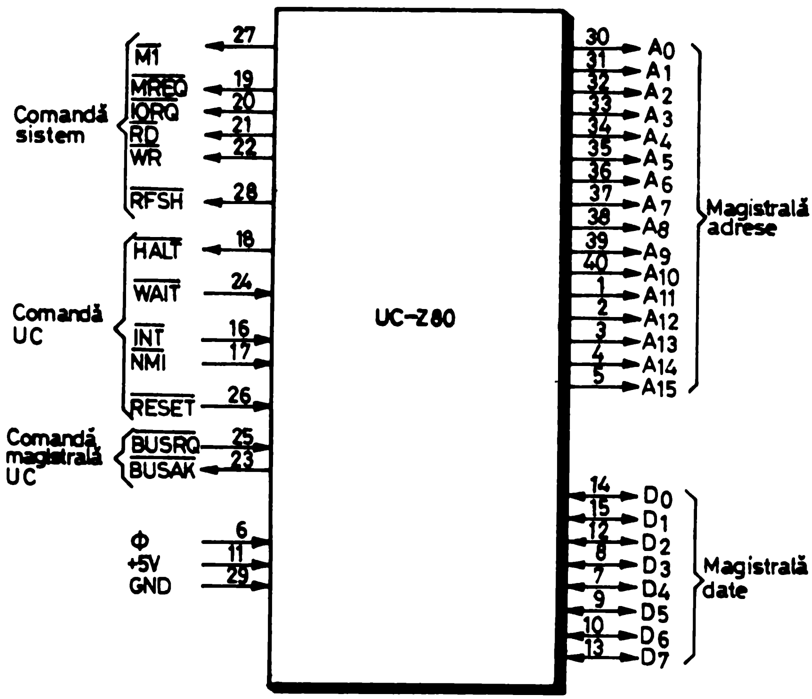


Fig. 2.4. Conexiunile externe ale UC-Z80

grupate pe funcţiuni: magistralele de adrese şi de date, semnalele pentru comanda sistemului, unităţii centrale şi magistralei UC-Z80, precum şi ceasul şi alimentarea microprocesorului. Vom descrie în continuare aceste semnale.

2.1.3.1. **Magistrala de adrese**

$A_0 \div A_{15}$, *Adress Bus*. Ieşiri trei-stări active pe "1". Magistrala de adrese de 16 biţi permite adresarea unei memorii externe de maximum 64 Kocteţi şi a intrării/ieşirii, I/E. Deoarece I/E se adresează cu cei mai puţin semnificativi 8 biţi, se pot selecta 256 *port*-uri de intrare, respectiv 256 *port*-uri de ieşire. Pe timpul acţiunii de reîmprospătare a memoriilor dinamice cei mai puţin semnificativi 7 biţi conţin adresa de împrospătare.

2.1.3.2. **Magistrala de date**

$D_0 \div D_7$, *Data Bus*. Intrări/ieşiri trei-stări active pe "1". Magistrala de date e utilizată pentru vehicularea bidirecţională a datelor între UC-Z80, memoria externă şi dispozitivele de I/E.

2.1.3.3. **Semnalele pentru comanda sistemului**

$\overline{M1}$, *Machine Cycle One*, primul ciclu-maşină. Ieşire activă pe "0". Indică, atunci când apare împreună cu semnalul $\overline{MREQ}$, primul ciclu al execuţiei unei instrucţiuni, ciclul de extragere din memorie a codului-operaţie. Dacă apare în conjuncţie cu $\overline{IORQ}$, semnifică începutul unui ciclu de achitare a întreruperii.

$\overline{MREQ}$, *Memory Request*, cerere la memorie. Ieşire trei-stări activă pe "0". Semnalează că pe magistrala de adrese s-a poziţionat o adresă ce va fi utilizată într-o operaţie de citire sau scriere în memoria externă.

$\overline{IORQ}$, *Input/Output Request*, cerere de I/E. Ieşire trei-stări activă pe "0". Indică poziţionarea pe octetul mai puţin semnificativ al magistralei $A_0 \div A_{15}$ a unei adrese ce va fi folosită într-o operaţie de citire sau scriere în dispozitivele de I/E. Dacă este generat împreună cu $\overline{M1}$ semnalează, pe timpul execuţiei unui ciclu de achitare a întreruperii, momentul când vectorul de răspuns poate fi plasat pe magistrala de date.

$\overline{RD}$, *Read*, citire. Ieşire trei-stări activă pe "0". Indică faptul că UC-Z80 vrea să citească date din memoria externă sau dintr-un dispozitiv de I/E. Memoria sau dispozitivul adresat utilizează acest semnal pentru a valida cu ajutorul lui poziţionarea datelor pe magistrala de date.

$\overline{WR}$, *Write*, scriere. Ieşire trei-stări activă pe "0". Semnalează faptul că pe magistrala de date se găsesc datele ce trebuie scrise în memoria externă sau dispozitivul periferic adresate.

$\overline{RFSH}$, *Refresh*, reîmprospătare. Ieşire activă pe "0". Împreună cu $\overline{MREQ}$ indică faptul că pe cei mai puţin semnificativi 7 biţi ai magistralei $A_0 \div A_{15}$ se găseşte o adresă de reîmprospătare ce poate fi utilizată de memoria dinamică a sistemului. Bitul A_7 este "0" iar pe $A_8 \div A_{15}$ se plasează conţinutul registrului I.

2.1.3.4. **Semnalele pentru comanda UC-Z80**

$\overline{HALT}$, *Halt State*, starea de oprire. Ieşire activă pe "0". Indică faptul că UC-Z80 a executat o instrucţiune de oprire şi este în aşteptarea unei întreruperi nemascabile sau a unei întreruperi mascabile validate în prealabil. Pe timpul cât e oprită, UC-Z80 execută instrucţiuni NOP pentru a asigura funcţia de reîmprospătare a memoriilor dinamice.

$\overline{WAIT}$, aşteptare. Intrare activă pe "0". Indică microprocesorului că memoria sau dispozitivele de I/E adresate nu sunt gata pentru a efectua un transfer de date. UC-Z80 rămâne în aşteptare cât timp semnalul $\overline{WAIT}$ este activ. Precizăm că pe durata cât semnalul $\overline{WAIT}$ este activ nu se generează $\overline{RFSH}$.

$\overline{INT}$, *Interrupt Request*, cerere de întrerupere. Intrare activă pe "0". Cerere de întrerupere generată de dispozitivele periferice. Cererea va fi achitată de UC-Z80 la sfârşitul instrucţiunii în curs, cu condiţia ca bistabilul de validare IFF1 să fi fost poziţionat în prealabil şi semnalul $\overline{BUSRQ}$ să nu fie activ. Achitarea în unul din cele trei moduri posibile se face cu un ciclu de achitare recunoscut prin aceea că se generează $\overline{IORQ}$ pe durata lui $\overline{M1}$. De obicei, la intrarea INT se conectează într-un SAU cablat mai multe circuite, ceea ce necesită legarea unei rezistenţe la +5V.

$\overline{NMI}$, *Non-Maskable Interrupt*, întrerupere nemascabilă. Intrare activă pe frontul negativ semnalând o cerere de întrerupere nemascabilă. Prioritară faţă de $\overline{INT}$, întreruperea nemascabilă va fi întotdeauna recunoscută la sfârşitul instrucţiunii curente, indiferent de starea bistabilului IFF1. $\overline{NMI}$ forţează o instrucţiune de restart la locaţia 0066H. Facem observaţiile că durata instrucţiunii curente se poate mări prin intrarea în stări WAIT şi că semnalul $\overline{BUSRQ}$ este prioritar faţă de $\overline{NMI}$.

$\overline{RESET}$, iniţializare. Intrare activă pe "0". Activând această intrare se iniţializează UC-Z80, adică:

- numărătorul de program se forţează pe zero;
- bistabilii IFF1 şi IFF2 se pun pe "0", invalidându-se întreruperile;
- registrele I şi R se fac 00H;
- se stabileşte modul 0 de tratare a întreruperilor.

Pe timpul iniţializării, magistralele de adrese şi date trec în starea de impedanţă mare, iar toate semnalele de comandă devin inactive. De asemenea, nu se generează semnale de reîmprospătare. Precizăm că pentru a se asigura iniţializarea completă a microprocesorului intrarea $\overline{RESET}$ trebuie să fie activă minimum trei cicli de ceas.

$\overline{BUSRQ}$, *Bus Request*, cerere de magistrală. Intrare activă pe "0". Intrarea este folosită pentru a solicita din exteriorul microprocesorului preluarea magistralelor de adrese şi de date, precum şi a ieşirilor trei-stări $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ şi $\overline{WR}$ astfel încât acestea să poată fi comandate de alte dispozitive. Semnalul $\overline{BUSRQ}$, prioritar faţă de $\overline{NMI}$, va fi întotdeauna achitat după terminarea ciclului-maşină în curs de execuţie, UC-Z80 trecând magistralele şi ieşirile trei-stări pe starea de impedanţă înaltă. Ca şi $\overline{INT}$, la intrarea $\overline{BUSRQ}$

pot fi conectate mai multe circuite într-un SAU cablat, ceea ce impune legarea unei rezistențe la +5V.

$\overline{\text{BUSAK}}$, *Bus Aknowledge*, achitare magistrală. Ieşire activă pe "0". Indică dispozitivului solicitant că magistralele de adrese şi date, precum şi ieşirile trei-stări ale UC-Z80, au fost comandate din exterior. Facem observația că, pe durata cât ieşirea $\overline{\text{BUSAK}}$ e activă, nu se generează semnale de reîmprospătare.

2.1.4. DIAGRAME DE TIMP

Execuția unei instrucțiuni a UC-Z80 este constituită dintr-o secvență specifică de operații ca, de exemplu, citire/scriere într-un dispozitiv de I/E sau achitare de întrerupere. Fiecare dintre aceste operații de bază poate dura 4÷6 perioade de ceas. Perioada ceasului Φ este denumită ciclu sau stare T (T_1, T_2, ...), iar operațiile de bază, cicli-maşină M (M_1, M_2, ...). Un ciclu-maşină M este format din mai multe stări T. Ciclii-maşină pot fi prelungiți prin inserarea de stări WAIT. Execuția unei instrucțiuni, un ciclu-instrucțiune, va însemna deci o succesiune de câțiva cicli-maşină executați la rândul lor în mai multe stări T (ca în figura 2.5). Primul ciclu-maşină al oricărei instrucțiuni este un ciclu de

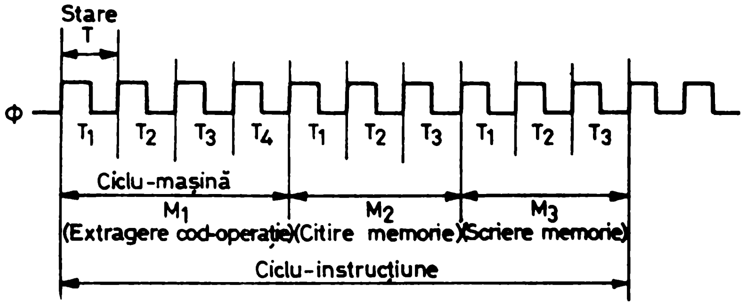


Fig. 2.5. Structura generală a unui ciclu-instrucțiune UC-Z80

extragere, *fetch*, a codului-operație al instrucțiunii ce urmează a fi executată. În următorii cicli UC-Z80 poate efectua transferuri de date cu memoria externă sau dispozitivele periferice. Vom prezenta în continuare desfăşurarea în timp a funcționării UC-Z80 pentru operațiile ce stau la baza execuției instrucțiunilor.

2.1.4.1. **Ciclul de extragere a codului-operație**

Diagrama de timp este dată în figura 2.6. La începutul ciclului M_1 microprocesorul transferă conținutul numărătorului de program PC pe magistrala de adrese. După aproximativ o jumătate de perioadă de ceas se activează semnalul $\overline{\text{MREQ}}$. În acest timp se consideră că liniile de adresă s-au stabilizat încât frontul negativ al lui $\overline{\text{MREQ}}$ poate fi utilizat direct pentru

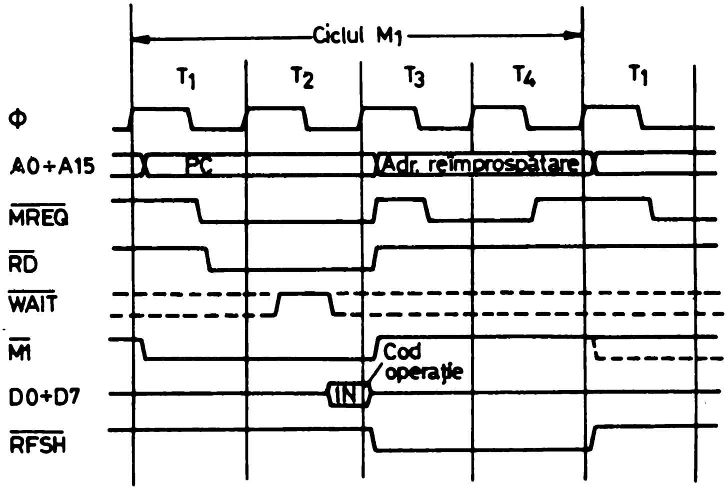


Fig. 2.6. Ciclul de extragere a codului-operaţie

validarea memoriilor dinamice. O dată cu $\overline{MREQ}$ se activează şi comanda de citire $\overline{RD}$ pentru a indica plasarea datelor citite din memorie pe magistrala de date a UC-Z80. Magistrala de date este eşantionată de microprocesor pe frontul pozitiv al ceasului în starea T_3. Totodată, pe acelaşi front, sunt dezactivate semnalele $\overline{MREQ}$ şi $\overline{RD}$. Stările T_3 şi T_4 sunt folosite de microprocesor pentru a realiza în paralel două funcţii: decodificarea şi execuţia instrucţiunii extrase, şi reîmprospătarea memoriei dinamice. În timpul acestor două stări cei mai puţin semnificativi 7 biţi ai magistralei de adrese reprezintă o adresă de reîmprospătare a memoriei. Activarea semnalului $\overline{RFSH}$ indică faptul că trebuie realizată o citire de reîmprospătare a tuturor memoriilor dinamice.

În figura 2.7 se prezintă un ciclu de extragere a codului-operaţie întârziat

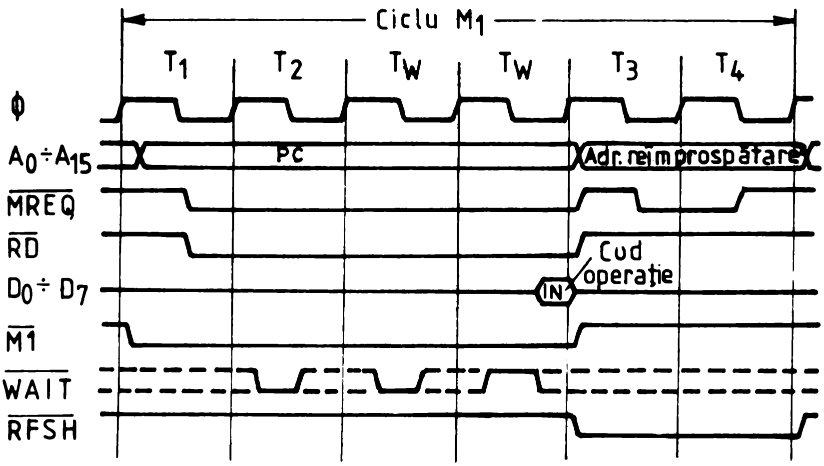


Fig. 2.7. Ciclul de extragere întârziat cu stări WAIT

datorită activării semnalului $\overline{WAIT}$. UC-Z80 eşantionează intrarea $\overline{WAIT}$, pe timpul stării T_2, cu frontul negativ al ceasului. Dacă este pe "0" se intră într-o stare de aşteptare T_w. În timpul fiecărei stări T_w pe frontul negativ al lui Φ, se investighează starea liniei $\overline{WAIT}$. Cât timp aceasta este "0", microprocesorul rămâne în starea T_w; la schimbarea ei pe "1" se trece în starea T_3, apoi T_4. Atragem din nou atenţia că pe timpul stărilor T_w nu se generează semnale de reîmprospătare, pentru aceasta fiind nevoie de două stări succesive, aşa cum sunt T_3 şi T_4. Întârzierea unei operaţii de citire din memoria externă prin activarea semnalului $\overline{WAIT}$ se foloseşte în cazurile când timpii de acces depăşesc, aproximativ, durata unei perioade de ceas şi, deci, stabilitatea datelor nu mai poate fi garantată la începutul stării T_3, pe frontul pozitiv al lui Φ.

2.1.4.2. Ciclii de citire şi scriere din/în memoria externă

În figura 2.8 se înfăţişează diagrama de timp a ciclilor de citire şi scriere din/în memorie diferiţi de ciclul de extragere în timpul căruia se genera $\overline{M1}$. Aceşti cicli durează trei stări, în afara cazurilor când se activează intrarea $\overline{WAIT}$. Se observă că semnalele $\overline{MREQ}$ şi $\overline{RD}$ se generează la fel ca într-un ciclu de extragere. Într-un ciclu de scriere, $\overline{MREQ}$ e activat după stabilizarea magistralei de adrese, iar $\overline{WR}$ după stabilizarea magistralei de date; acesta din urmă poate fi folosit direct ca impuls de citire/scriere pentru majoritatea memoriilor.

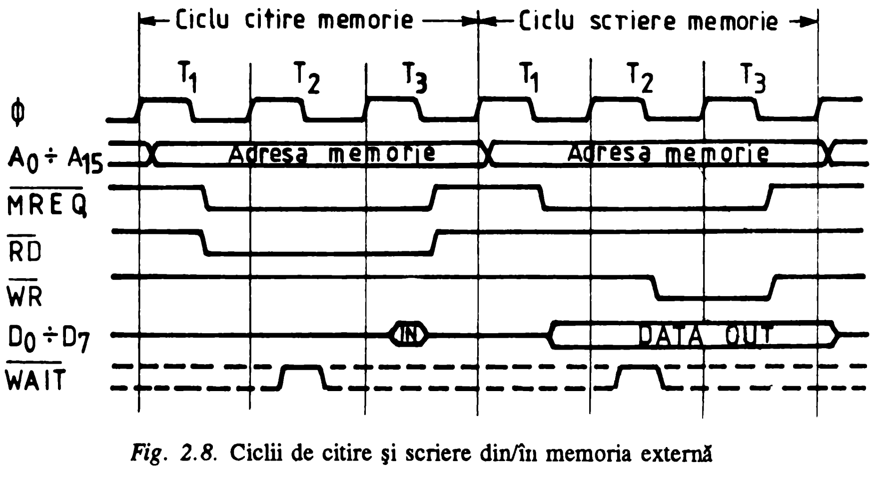


Fig. 2.8. Ciclii de citire şi scriere din/în memoria externă

Ciclii de citire sau scriere întârziaţi datorită activării semnalului $\overline{WAIT}$ pe durata frontului negativ al ceasului stării T_2 au o desfăşurare în timp ca în figura 2.9.

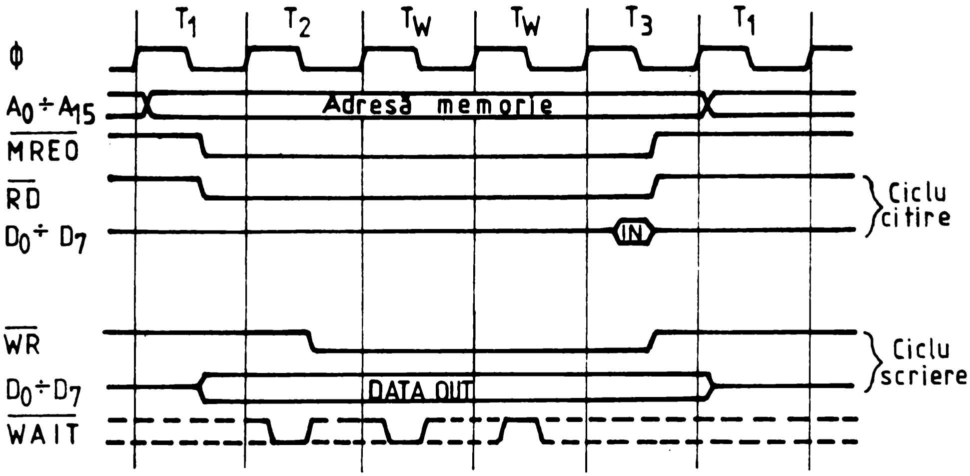


Fig. 2.9. Ciclii de citire şi scriere întârziaţi cu stări WAIT

2.1.4.3. Ciclii de intrare şi ieşire

Figura 2.10 ilustrează modul de funcţionare al microprocesorului Z80 în timpul operaţiilor de intrare/ieşire. Pe durata acestor operaţii, UC-Z80 introduce în mod automat o stare T_w, după T_2,pentru a permite circuitului periferic adresat

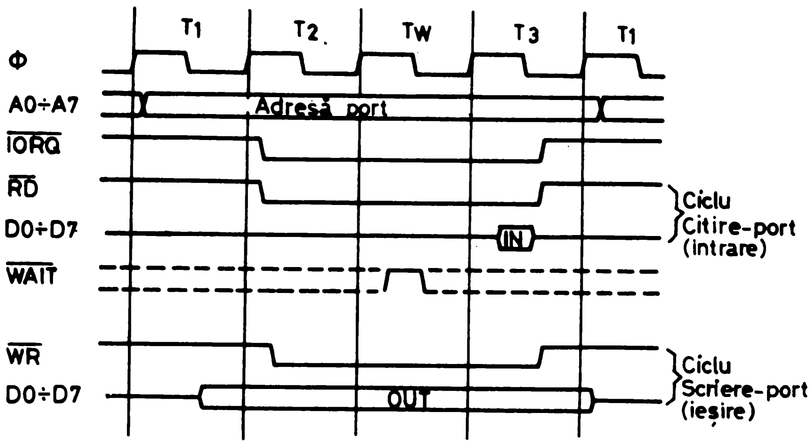


Fig. 2.10. Ciclii de intrare şi ieşire

să-şi decodifice adresa. Semnalul $\overline{WAIT}$ este testat pe timpul acestei stări T_w, în cazul în care este găsit activ menţinându-se starea de aşteptare. În figura 2.11 se arată diagrama de timp pentru ciclii de I/E cu mai mult de o stare T_w.

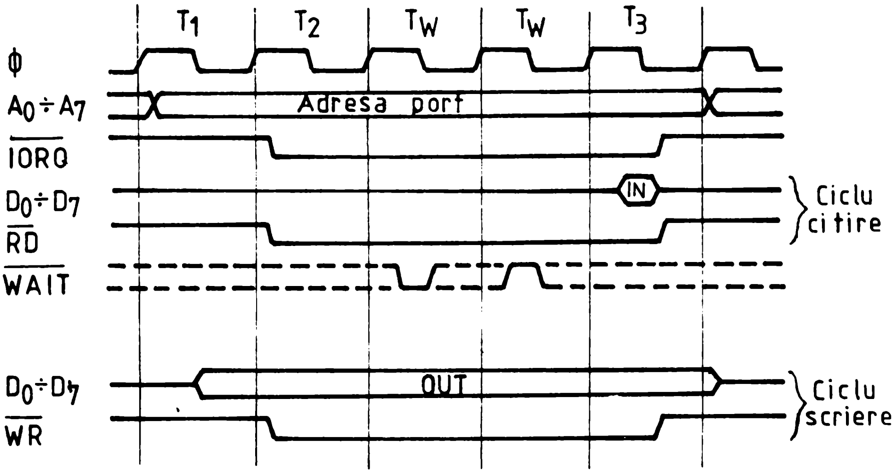


Fig. 2.11. Ciclii de intrare şi ieşire întârziaţi cu stări WAIT

2.1.4.4. Ciclul de cerere-achitare de magistrală

UC-Z80 eşantionează semnalul de cerere de magistrală, $\overline{BUSRQ}$, cu frontul pozitiv al ceasului, în ultima stare T a fiecărui ciclu-maşină (vezi figura 2.12). Dacă $\overline{BUSRQ}$ este "0", pe frontul pozitiv al următoarei stări T_x, microprocesorul va trece magistralele de adrese şi date precum şi ieşirile $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ şi $\overline{WR}$ în starea de impedanţă mare, *tri-state*, activând totodată şi semnalul de achitare $\overline{BUSAK}$. Pe timpul stărilor T_x, cât aceste linii

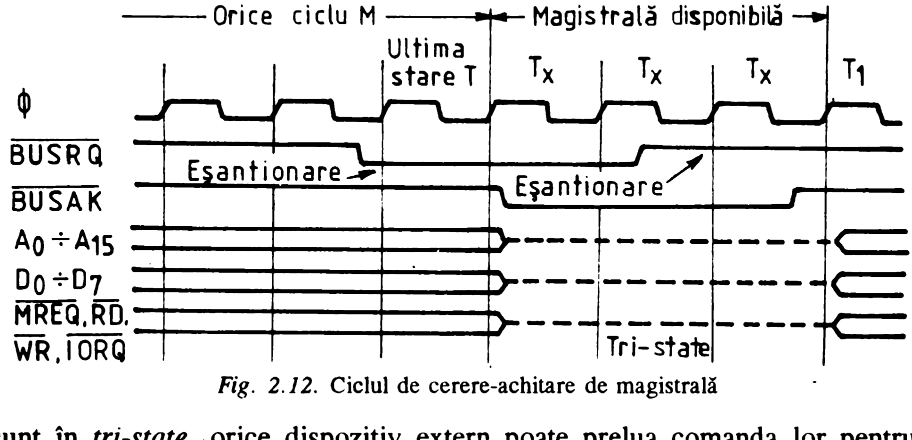


Fig. 2.12. Ciclul de cerere-achitare de magistrală

sunt în *tri-state*, orice dispozitiv extern poate prelua comanda lor pentru a efectua, de obicei, transferuri directe de date între memorie şi periferice (aşa-numitele transferuri cu acces direct la memorie, DMA). Observăm, aşadar, că întârzierea maximă de răspuns a microprocesorului la o cerere de magistrală este de un ciclu-maşină, dispozitivul extern putând să menţină controlul magistralelor oricât de mult. Atragem atenţia, însă, că pe durata unor transferuri

DMA, $\overline{BUSAK}$ = 0, UC-Z80 nu efectuează reîmprospătarea memoriei dinamice. Pentru a asigura funcţionarea fără eroare a memoriilor dinamice, în special în cazul transferurilor de blocuri mari, funcţia de reîmprospătare trebuie preluată de dispozitivele externe. Menţionăm, de asemenea, că în timpul unui ciclu de cerere-achitare de magistrală activitatea UC-Z80 nu poate fi suspendată de nici o întrerupere, $\overline{INT}$ sau $\overline{NMI}$.

2.1.4.5. **Ciclul de cerere-achitare a întreruperii mascabile**

Un ciclu de cerere-achitare a întreruperii mascabile se desfăşoară ca în figura 2.13. La sfârşitul fiecărei instrucţiuni, microprocesorul eşantionează semnalul de întrerupere mascabilă, $\overline{INT}$, cu frontul pozitiv al ultimei stări T. Semnalul $\overline{INT}$ activ va fi acceptat numai dacă bistabilul intern de validare, IFF1, comandat prin software, este pus pe "1" şi dacă semnalul de cerere de

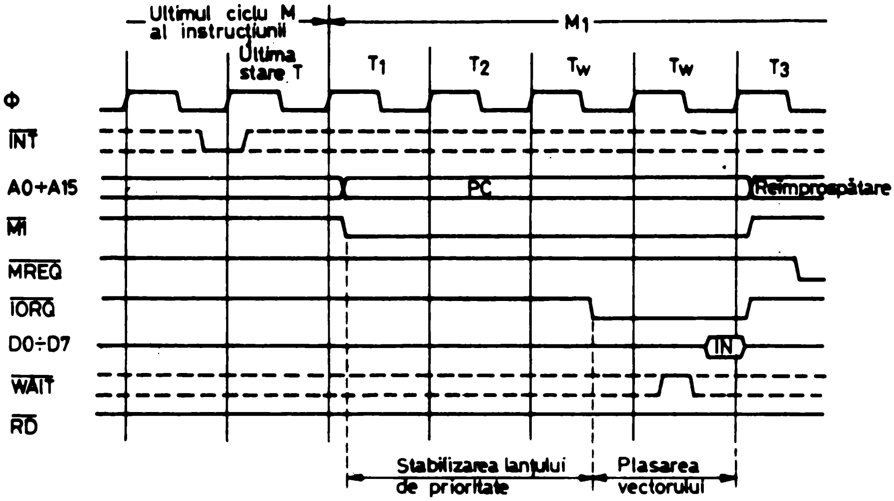


Fig. 2.13. Ciclul de cerere-achitare a întreruperii mascabile

magistrală, $\overline{BUSRQ}$, nu este "0". Dacă întreruperea se acceptă, în continuare va fi declanşat un ciclu M_1 special: în timpul acestui ciclu, după poziţionarea semnalului $\overline{M1}$ pe "0" se activează, în locul lui $\overline{MREQ}$ – cum se făcea într-un ciclu obişnuit de extragere –, $\overline{IORQ}$. Astfel se indică, într-un mod unic, dispozitivului periferic care întrerupe, să plaseze un vector de 8 biţi pe magistrala de date.

În figura 2.13 se observă introducerea automată, de către microprocesor, a două stări de aşteptare. Dispozitivele de I/E din seria Z80 pot fi conectate într-un lanţ de priorităţi cu ajutorul semnalelor de intrare/ieşire IEI/IEO (I*nterrupt* E*nable* I*n*, I*nterrupt* E*nable* O*ut*). Într-o asemenea conectare,

prioritatea unui dispozitiv e dată de poziţia fizică pe care o ocupă în lanţ. Pe timpul unui ciclu de întrerupere, la activarea lui $\overline{M1}$ starea priorităţilor se îngheaţă, propagarea semnalelor de prioritate IEI/IEO de-a lungul lanţului în care sunt conectate dispozitivele periferice trebuind să se stabilizeze până la generarea lui $\overline{IORQ}$. Tocmai pentru a permite acest lucru s-au inserat cele două stări T_w. Întârzierea asigură stabilizarea semnalelor de prioritate şi, în plus, autoidentificarea dispozitivului de I/E ce va plasa vectorul pe magistrala de date.

Într-un sistem Z80 ce lucrează în întreruperi orice dispozitiv solicitant poate pune linia $\overline{INT}$ pe "0". Dacă semnalul de prioritate IEI, care intră într-un dispozitiv ce a activat linia $\overline{INT}$, este "1", dispozitivul îşi va plasa vectorul său de întrerupere pe magistrala de date la trecerea lui $\overline{IORQ}$ pe "0". După aceasta, perifericul va elibera linia $\overline{INT}$ pentru a permite întreruperi de la dispozitivele prioritare. Dispozitivele cu prioritate scăzută sunt inhibate, deoarece semnalul de prioritate emis, IEO, este menţinut pe "0". Semnalul IEO va fi pus pe "1" de către perifericul activ (cel care are IEI=1 şi IEO=0), la decodificarea instrucţiunii de revenire RETI, adică la sfârşitul subrutinei de serviciu. Poziţionarea lui IEO pe "1" validează achitarea întreruperilor şi pentru dispozitivele cu prioritate scăzută.

Microprocesorul Z80 poate fi programat să răspundă la întreruperea mascabilă în trei moduri posibile.

În modul 0, identic cu modul de funcţionare al lui 8080, dispozitivul periferic este cel care generează instrucţiunea următoare. În acest caz ciclul de cerere-achitare al întreruperii este echivalent cu un ciclu de extragere, iar timpul necesar execuţiei instrucţiunii inserate de periferic e mai lung cu doi cicli decât atunci când codul-operaţie se citeşte din memorie. Cei doi cicli se datorează introducerii stărilor de aşteptare, T_w, necesare stabilizării semnalelor de prioritate IEI/IEO. Aceeaşi mărire cu doi cicli a duratei de execuţie se produce şi în cazul când UC-Z80 este programată să lucreze în modul 1 când, ca răspuns la o întrerupere, se execută o instrucţiune de restart la locaţia 0038H.

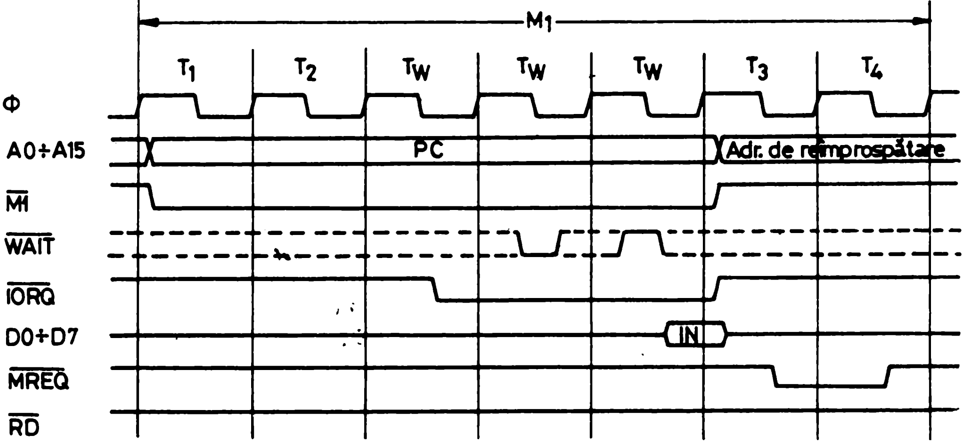


Fig. 2.14. Ciclul de cerere-achitare a întreruperii cu stări WAIT

În situaţia în care se lucrează în modul 2, după ce dispozitivul de I/E poziţionează vectorul, porţiunea cea mai puţin semnificativă a *pointer*-ului, Z80 salvează automat în stivă numărătorul de program, obţine adresa de început a subrutinei de serviciu şi execută un salt la această adresă. Pentru a realiza operaţiile menţionate, UC-Z80 are nevoie de 19 cicli: 7 pentru a citi vectorul de la periferic, 6 pentru a salva numărătorul de programe şi 6 pentru a obţine adresa de salt.

În figura 2.14 se dă diagrama de timp a unui ciclu de cerere-achitare a întreruperii a cărui durată este mărită datorită activării de către periferic a liniei $\overline{WAIT}$.

2.1.4.6. Ciclul de cerere a întreruperii nemascabile

Figura 2.15 descrie modul de funcţionare a lui Z80 în cazul apariţiei unei întreruperi $\overline{NMI}$. $\overline{NMI}$ se eşantionează în acelaşi timp cu $\overline{INT}$, cu frontul pozitiv al ceasului, în ultima stare T a instrucţiunii în curs de execuţie. $\overline{NMI}$ este însă prioritar faţă de $\overline{INT}$ şi, de asemenea, nu poate fi mascat prin program. Un impuls pe linia $\overline{NMI}$ va conduce la poziţionarea unui bistabil intern, acesta fiind de fapt testat la sfârşitul fiecărei instrucţiuni. Răspunsul microprocesorului la $\overline{NMI}$ este asemănător cu un ciclu de citire din memorie. Diferenţa constă în aceea că, în situaţia de faţă, UC-Z80 ignoră codul plasat de memorie pe magistrala de date şi execută o instrucţiune de restart la 0066H.

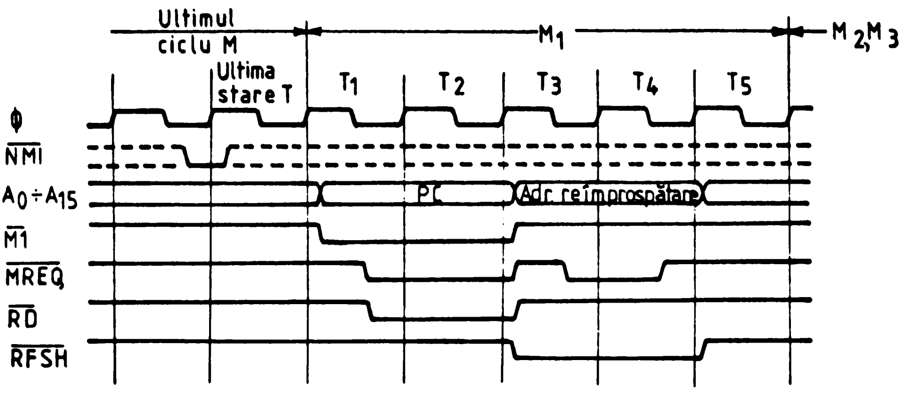


Fig. 2.15. Ciclul de cerere-achitare a întreruperii nemascabile

Pentru a înţelege mai bine relaţia între cele trei semnale, $\overline{BUSRQ}$, $\overline{NMI}$ şi $\overline{INT}$, ce pot întrerupe fluxul normal al execuţiei instrucţiunilor, prezentăm în continuare organigrama din figura 2.16. Aşa cum se vede, $\overline{INT}$ şi $\overline{NMI}$ sunt luate în consideraţie la sfârşitul instrucţiunii, în timp ce $\overline{BUSRQ}$ la sfârşitul unui ciclu-maşină. Ordinea de prioritate a acestor semnale, începând cu cel prioritar este: $\overline{BUSRQ}$, $\overline{NMI}$ şi $\overline{INT}$. În timp ce UC-Z80 este în modul

DMA, cu magistralele comandate de un periferic, nu se răspunde la $\overline{NMI}$ sau $\overline{INT}$.

Desfăşurarea pe cicli-maşină a răspunsurilor UC-Z80 la întreruperi este dată în tabelul 2.1.

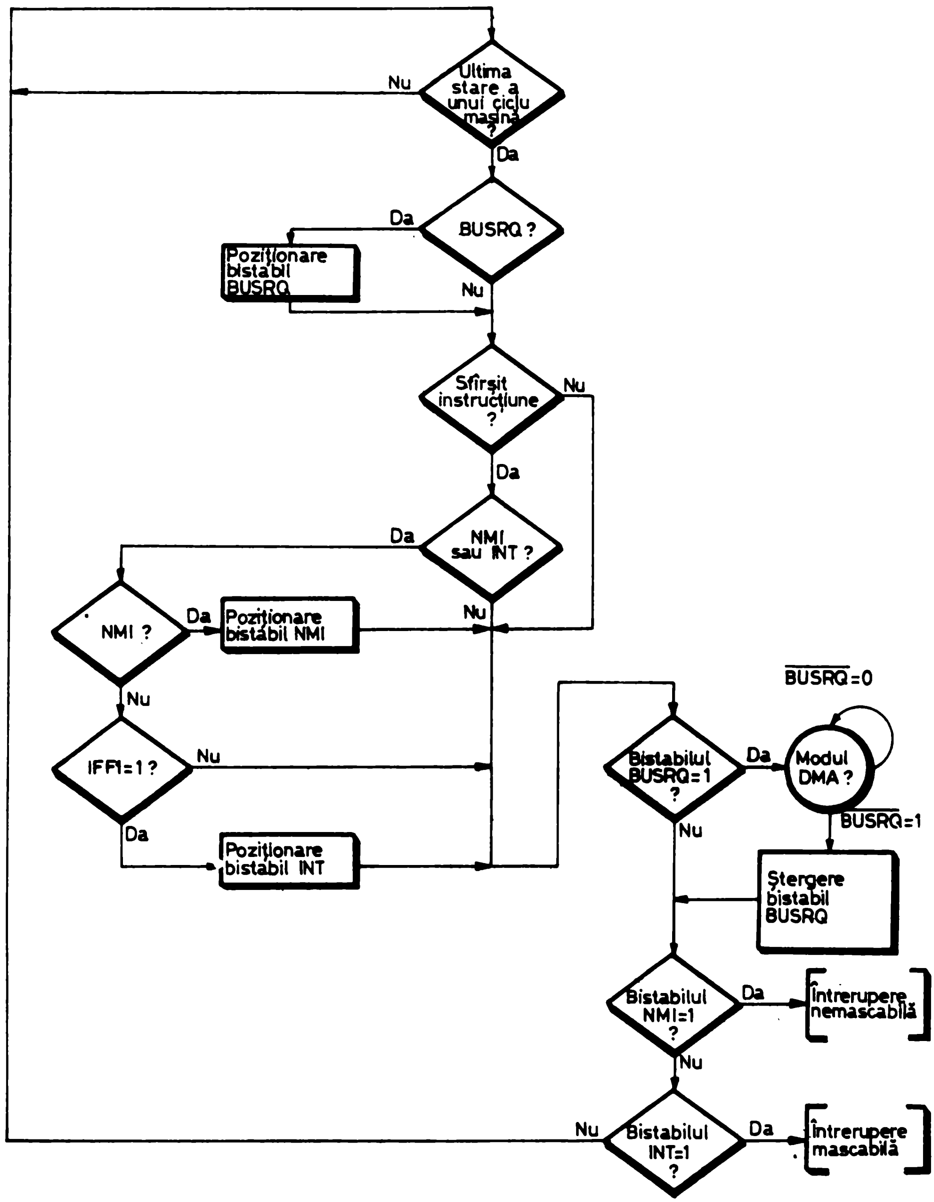


Fig. 2.16. Relaţia intre $\overline{BUSRQ}$, $\overline{NMI}$ şi $\overline{INT}$

TABELUL 2.1. **Răspunsuri la întreruperi**

Operația	Ciclii-mașină					Observații
	M_1	M_2	M_3	M_4	M_5	
Întrerupere nemascabilă	ECO(5) se ignoră codul-operație SP−1 →	SSH(3) SP−1 →	SSL(3)			
Întrerupere mascabilă *Modul 0*	INTA(6)	COL(3)	COH(4) SP−1 →	SSH(3) SP−1 →	SSL(3)	Perifericul generează instrucțiunea "CALL"
	INTA(6) SP−1 →	SSH(3) SP−1 →	SSL(3)			Perifericul generează instrucțiunea "RST"
Întrerupere mascabilă *Modul 1*	INTA(7) SP−1 →	SSH(3) SP−1 →	SSL(3)			RST 38H generat intern
Întrerupere mascabilă *Modul 2*	INTA(7) SP−1 →	SSH(3) SP−1 →	SSL(3)	CML(3)	CMH(3)	Perifericul generează în M_1 vectorul de întrerupere

Notă: ECO – Extragere cod operație
SSH – Scriere în stivă – octet mai semnificativ
SSL – Scriere în stivă – octet mai puțin semnificativ
INTA – Ciclu de cerere-achitare a întreruperii
COL – Citire operand – octet mai puțin semnificativ
COH – Citire operand – octet mai semnificativ
CML – Citire din memoria externă – octet mai puțin semnificativ
CMH – Citire din memoria externă – octet mai semnificativ
( ) – Număr de stări T necesare execuției unui ciclu-mașină

2.1.4.7. Ciclul de ieşire din starea HALT

Apariția unei instrucțiuni HALT va conduce la execuția de către UC-Z80 de NOP-uri. Ieșirea $\overline{HALT}$, pusă pe "0" la începutul secvenței de NOP-uri, rămâne activă până la primirea unei întreruperi, $\overline{NMI}$ sau $\overline{INT}$, aceasta din urmă cu condiția să fi fost validată în prealabil.

Microprocesorul execută NOP-uri în starea HALT pentru a asigura funcția de reîmprospătare a memoriilor dinamice. Aici un NOP este de fapt un ciclu de extragere în care UC-Z80 va ignora codul de pe magistrala de date înlocuindu-l intern cu instrucțiunea NOP.

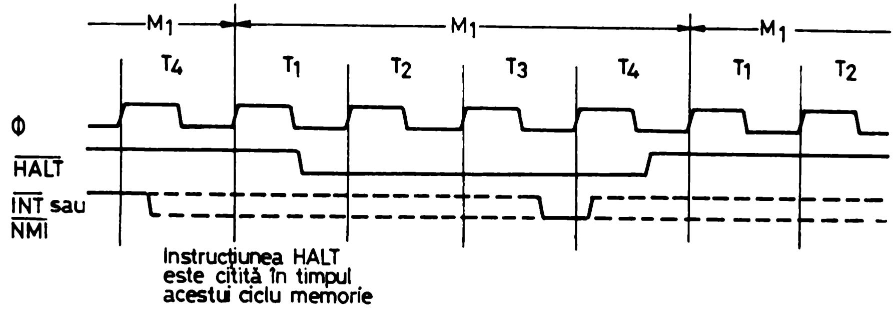


Fig. 2.17. Ciclul de ieşire din starea HALT

Cele două linii de întrerupere sunt eşantionate cu frontul pozitiv al ceasului în starea T_4 a fiecărui NOP, aşa cum se poate vedea în figura 2.17. Dacă unul din semnalele $\overline{INT}$ sau $\overline{NMI}$ a fost găsit activ se iese din starea HALT pe următorul front pozitiv şi se intră într-un ciclu de achitare corespunzător tipului de întrerupere şi modului de lucru programat.

2.1.4.8. **Ciclul de inițializare**

Inițializarea unității centrale UC-Z80 se face cu ajutorul semnalului $\overline{RESET}$. Acest semnal trebuie să fie activ cel puțin trei perioade de ceas pentru ca UC-Z80 să îl accepte în mod corect. Pe durata cât $\overline{RESET}$ rămâne activ magistralele de adrese şi date sunt în starea de impedanță înaltă, iar ieşirile de comandă sunt inactive. În figura 2.18 se dă diagrama unui ciclu de inițializare. După ce $\overline{RESET}$ devine inactiv, sunt necesari încă trei cicli T înainte ca microprocesorul să-şi reia modul de funcționare normal. La inițializare numărătorul de program se şterge încât în primul ciclu de extragere se va citi codul-operație de la adresa 0000H.

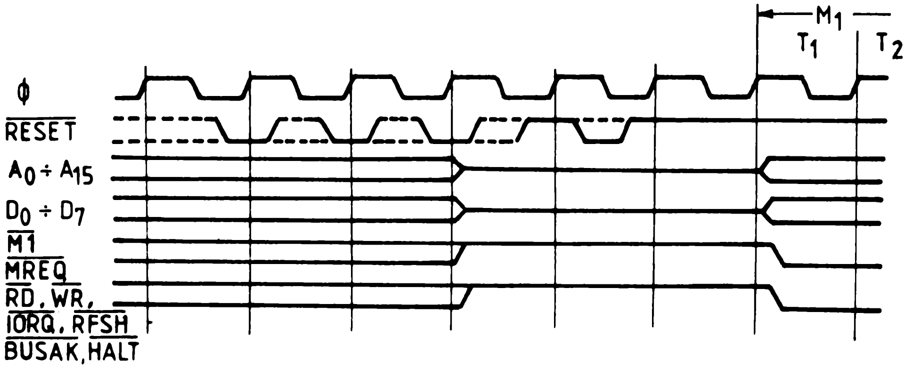


Fig. 2.18. Ciclul de inițializare

2.1.5. SETUL DE INSTRUCŢIUNI

Microprocesorul Z80 are unul dintre cele mai puternice şi versatile seturi de instrucţiuni în raport cu celelalte microprocesoare pe 8 biţi. El poate executa 158 de tipuri diferite de instrucţiuni, incluzând printre ele şi setul de 78 instrucţiuni al lui 8080 (în cod obiect). Ca exemple de instrucţiuni foarte puternice putem da instrucţiunile speciale de transfer de blocuri sau instrucţiunile de prelucrare pe bit.

Setul de instrucţiuni Z80 va fi prezentat în continuare sub forma unor tabele sintetice, indicându-se mnemonica instrucţiunii în limbaj de asamblare Z80, descrierea simbolică a operaţiei, codul-obiect al instrucţiunii, starea indicatorilor de condiţii, desfăşurarea pe cicli-maşină a instrucţiunii. Instrucţiunile compatibile 8080 au mnemonica în limbaj de asamblare 8080 trecută într-o coloană separată.

Instrucţiunile Z80 se pot împărţi în mai multe categorii:

- transferuri pe 8 biţi;
- transferuri pe 16 biţi;
- schimburi între registre, transfer de blocuri, căutări;
- operaţii aritmetice şi logice pe 8 biţi;
- operaţii aritmetice generale şi operaţii de comandă a UC-Z80;
- operaţii aritmetice pe 16 biţi;
- rotaţii şi deplasări;
- prelucrări pe bit;
- salturi;
- apeluri de subrutine, reveniri, instrucţiuni de restart;
- operaţii de I/E.

Modurile de adresare implementate în UC-Z80 permit un transfer eficient de date între registre, memoria externă şi dispozitivele periferice. Vom descrie mai jos, în mod succint, modurile de adresare ale microprocesorului, adică mecanismele de generare a adresei datelor necesare în execuţia fiecărei instrucţiuni.

Adresare imediată

Cod-operaţie	} 1 sau 2 octeţi
Operand	

Octetul care se găseşte în memorie după codul-operaţie reprezintă operandul necesar.

Adresare imediată extinsă

Cod-operaţie	} 1 sau 2 octeţi
Operand	
Operand	

Cei doi octeţi ce se găsesc în memorie după codul-operaţie reprezintă operandul necesar.

Adresare directă în pagina zero

Cod-operaţie	} 1 octet

b_7 b_0

Adresa efectivă: $00b_5b_4b_3000$

Cu ajutorul unei instrucţiuni de restart, de un octet, se poate specifica adresa completă a unei locaţii din pagina zero. De obicei, la această locaţie se plasează începutul unei subrutine, apelarea ei necesitând deci un singur octet.

Adresare relativă:

Cod-operaţie	} 1 octet
Deplasament	

Octetul care se găseşte în memorie după codul-operaţie reprezintă un *deplasament*. Acest deplasament, un număr de 8 biţi cu semn, scris în complement faţă de 2, se adună la adresa instrucţiunii următoare.

Adresare extinsă

Cod-operaţie	} 1 sau 2 octeţi
Octetul mai puţin semnificativ al adresei de salt sau de operand	
Octetul mai semnificativ al adresei de salt sau de operand	

Cei doi octeţi care urmează codului-operaţie constituie o adresă de salt sau o adresă de operand.

Adresare indexată

Cod-operaţie	} 2 octeţi
Cod-operaţie	
Deplasament	

Octetul ce urmează codului-operaţie este un deplasament ce se adună la conţinutul unuia dintre cele două registre index pentru a forma o adresă de memorie. Acest deplasament este, ca şi în cazul adresării relative, un număr cu semn reprezentat în complement faţă de 2 (-128, $+127$).

Adresare de registru. Codul-operaţie specifică explicit registrele utilizate de instrucţiune.

Adresare implicită. Codul-operaţie specifică implicit registrele utilizate, de exemplu, acumulatorul este întotdeauna registru-destinaţie în operaţiile aritmetice.

Adresare indirectă cu registre. Codul-operaţie precizează un registru pereche, de exemplu HL, BC, care conţine adresa de memorie.
Adresare pe bit. Codul-operaţie al unei instrucţiuni de prelucrare pe bit specifică bitul asupra căruia se execută operaţia. Octetul din care face parte bitul respectiv poate fi adresat, la rândul lui, cu ajutorul adresării de registru, indirecte cu registru sau indexate.

Menţionăm că unele din instrucţiunile Z80, de exemplu cele aritmetice sau transferurile, pot avea mai mulţi operanzi. În aceste cazuri se folosesc două tipuri de adresare. De pildă, un transfer utilizează adresarea imediată pentru a specifica sursa şi adresarea indirectă cu registru sau indexată pentru a specifica destinaţia.

În tabelele prezentate mai jos se vor folosi următoarele notaţii şi simboluri pentru indicatorii de condiţii:

S — *Indicator de semn.* S = 1 dacă cel mai semnificativ bit al rezultatului este "1", rezultatul este negativ.

Z — *Indicator de zero.* Z = 1 dacă rezultatul operaţie este o încărcare a lui 00H în acumulator.

P/V — *Indicator de paritate/depăşire.* După cum se vede, paritatea (P) şi depăşirea utilizează acelaşi indicator de condiţie. Operaţiile logice afectează indicatorul în funcţie de paritatea rezultatului, iar cele aritmetice în funcţie de depăşire. Dacă P/V memorează paritatea atunci P/V = 1 când rezultatul este par. În cazul în care P/V memorează depăşirea atunci indicatorul este "1" când rezultatul produce depăşire. Indicatorul de depăşire, V, indică faptul că numărul în complement faţă de 2 din acumulator este eronat deoarece s-a depăşit gama permisă: maximum +127, minimum −128. De exemplu:

```
        +119 =   0111 0111
       +   9 =   0000 1001
       -----     ---------
       C = 0     1000 0000 = -128 (greşit). Apare depăşire.
```

În acest caz rezultatul este incorect: apare depăşire iar indicatorul de transport rămâne "0". Situaţia va fi sesizată de indicatorul V care se va poziţiona pe "1".

H — *Indicator de transport auxiliar.* H = 1 dacă operaţiile de adunare sau scădere au produs un transport sau un împrumut din bitul 4 al acumulatorului. Indicatorul H nu poate fi testat prin program el fiind utilizat numai de instrucţiunea DAA.

N — *Indicator de adunare/scădere.* N = 1 dacă operaţia precedentă a fost scădere. De asemenea, nu poate fi testat prin program fiind utilizat numai de DAA.

C — *Indicator de transport.* C = 1 dacă operaţia a produs un transport din cel mai semnificativ bit al operandului sau al rezultatului.

↕ — Indicatorul este afectat în funcţie de rezultatul operaţiei.

• — Indicatorul nu este modificat de operaţie.

0 — Indicatorul este pus pe "0" de operaţie.

1 — Indicatorul este pus pe "1" de operaţie.

× Valoarea indicatorului nu are importanță.
V P/V memorează depășirea.
P P/V memorează paritatea.

În scrierea mnemonicelor Z80 se vor întrebuința următoarele notații:

r, *r'* Unul din registrele UC-Z80: A, B, C, D, E, H, L.
n Expresie pe un octet în gama (0,255).
(HL) Conținutul memoriei la locația adresată de conținutul HL.
d Expresie pe un octet în gama (−128, +127).
nn Expresie pe doi octeți în gama (0, 65535).
(*nn*) Conținutul memoriei la locația adresată de expresia *nn*.
dd Unul din registrele duble: BC, DE, HL, SP.
qq Unul din registrele duble: BC, DE, HL, AF.
s Poate fi *r*, *n*, (HL), (IX+*d*) sau (IY+*d*).
m Poate fi *r*, (HL), (IX+*d*) sau (IY+*d*).
ss Unul din registrele duble: BC, DE, HL, SP.
pp Unul din registrele duble: BC, DE, IX, SP.
rr Unul din registrele duble: BC, IY, SP, DE.
b Expresie în gama (0, 7).
cc Starea indicatorilor de condiții.
e Expresie pe un octet în gama (−126, +129).

Pentru mnemonicele 8080 se mai folosesc și simbolurile:

rp Unul din registrele pereche; B pentru BC, D pentru DE, H pentru HL și SP pentru indicatorul vârfului de stivă.
PSW Desemnează perechea de registre AF.

Descrierea ciclilor-mașină se va face utilizând următoarele prescurtări:

CM Citire din memoria externă.
CMH Citire din memoria externă – octet mai semnificativ.
CML Citire din memoria externă – octet mai puțin semnificativ.
CO Citire operand.
COH Citire operand – octet mai semnificativ.
COL Citire operand octet mai puțin semnificativ.
CPO Citire din *port* de I/E.
CSH Citire din stivă – octet mai semnificativ.
CSL Citire din stivă – octet mai puțin semnificativ.
ECO Extragere cod-operație.
OI Operație internă.
SM Scriere în memoria externă.
SMH Scriere în memoria externă – octet mai semnificativ.
SML Scriere în memoria externă – octet mai puțin semnificativ.
SPO Scriere în *port* de I/E.
SSH Scriere în stivă – octet mai semnificativ.
SSL Scriere în stivă – octet mai puțin semnificativ.
( ) Număr de stări T necesare execuției unui ciclu-mașină.

TABELUL 2.2. **Transferuri pe 8 biţi**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiţii
			Binar	Hexa	S Z H P/V N C
LD r,r'	MOV r,r'	r ← r'	01 r r'		. × . × .
LD r,n	MVI r,n	r ← n	00 r 110 ← n →		. × . × .
LD r,(HL)	MOV r,M	r ← (HL)	01 r 110		. . × . × . . .
LD r,(IX+d)		r ← (IX+d)	11 011 101 01 r 110 ← d →	DD	. × . × . . .
LD r,(IY+d)		r ← (IY+d)	11 111 101 01 r 110 ← d →	FD	. × . × . .
LD (HL),r	MOV M,r	(HL) ← r	01 110 r		. . × . × . . .
LD (IX+d),r		(IX+d) ← r	11 011 101 01 110 r ← d →	DD	. . × . × .
LD (IY+d),r		(IY+d) ← r	11 111 101 01 110 r ← d →	FD	. × . × .
LD (HL),n	MVI M,n	(HL) ← n	00 110 110 ← n →	36	. × . × . .
LD (IX+d),n		(IX+d) ← n	11 011 101 00 110 110 ← d → ← n →	DD 36	. × . × . .
LD (IY+d),n		(IY+d) ← n	11 111 101 00 110 110 ← d → ← n →	FD 36	. × . × . .
LD A,(BC)	LDAX B	A ← (BC)	00 001 010	0A	. . × . × . . .
LD A,(DE)	LDAX D	A ← (DE)	00 011 010	1A	. . × . × . . .
LD A,(nn)	LDA nn	A ← (nn)	00 111 010 ← n → ← n →	3A	. × . × .
LD (BC),A	STAX B	(BC) ← A	00 000 010	02	. . × . × . .
LD (DE),A	STAX D	(DE) ← A	00 010 010	12	. . × . × . . .
LD (nn),A	STA nn	(nn) ← A	00 110 010 ← n → ← n →	32	. × . × .

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)					r, r' – Registru 000 – B 001 – C 010 – D 011 – E 100 – H 101 – L 111 – A
ECO(4)	CO(3)				
ECO(4)	CM(3)				
ECO(4)+ ECO(4)	CO(3)	OI(5)	CM(3)		
ECO(4)+ ECO(4)	CO(3)	OI(5)	CM(3)		
ECO(4)	SM(3)				
ECO(4)+ ECO(4)	CO(3)	OI(5)	SM(3)		
ECO(4)+ ECO(4)	CO(3)	OI(5)	SM(3)		
ECO(4)	CO(3)	SM(3)			
ECO(4)+ ECO(4)	CO(3)	CO(5)	SM(3)		
ECO(4)+ ECO(4)	CO(3)	CO(5)	SM(3)		Nota 1
ECO(4)	CM(3)				
ECO(4)	CM(3)				
ECO(4)	COL(3)	COH(4)	CM(3)		
ECO(4)	SM(3)				
ECO(4)	SM(3)				
ECO(4)	COL(3)	COH(3)	SM(3)		

TABELUL 2.2 **Transferuri pe 8 biţi (continuare)**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect: Binar	Codul obiect: Hexa	Indicatorii de condiţii: S Z H P/V N C
LD A,I		A ← I	11 101 101 01 010 111	ED 57	↕ ↕ × 0 × IFF 0 .
LD A,R		A ← R	11 101 101 01 011 111	ED 5F	↕ ↕ × 0 × IFF 0
LD I,A		I ← A	11 101 101 01 000 111	ED 47	. × . × .
LD R,A		R ← A	11 101 101 01 001 111	ED 4F	. × × .

TABELUL 2.3. **Transferuri pe 16 biţi**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect: Binar	Codul obiect: Hexa	Indicatorii de condiţii: S Z H P/V N C
LD dd,nn	LXI rp,nn	dd ← nn	00 dd0 001 ← n → ← n →		. . × . × . . .
LD IX,nn		IX ← nn	11 011 101 00 100 001 ← n → ← n →	DD 21	. × . × . . .
LD IY,nn		IY ← nn	11 111 101 00 100 001 ← n → ← n →	FD 21	. × × . .
LD HL,(nn)	LHLD nn	H ← (nn+1) L ← (nn)	00 101 010 ← n → ← n →	2A	. . × . × . . .
LD dd,(nn)		dd_H ← (nn+1) dd_L ← (nn)	11 101 101 01 dd1 011 ← n → ← n →	ED	. . × . × . . .
LD IX,(nn)		IX_H ← (nn+1) IX_L ← (nn)	11 011 101 00 101 010 ← n → ← n →	DD 2A	. . × . × . . .
LD IY,(nn)		IY_H ← (nn+1) IY_L ← (nn)	11 111 101 00 101 010 ← n → ← n →	FD 2A	. . × . × . . .

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)+ ECO(5)					
ECO(4)+ ECO(5)					
ECO(4)+ ECO(5)					
ECO(4)+ ECO(5)					

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)	COL(3)	COH(3)			dd – Registru 00 – BC 01 – DE 10 – HL 11 – SP
ECO(4)+ ECO(4)	COL(3)	COH(3)			Nota 2
ECO(4)+ ECO(4)	COL(3)	COH(3)			
ECO(4)	COL(3)	COH(3)	CML(3)	CMH(3)	
ECO(4)+ ECO(4)	COL(3)	COH(3)	CML(3)	CMH(3)	
ECO(4)+ ECO(4)	COL(3)	COH(3)	CML(3)	CMH(3)	
ECO(4)+ ECO(4)	COL(3)	COH(3)	CML(3)	CMH(3)	Indice H = mai semnificativ Indice L = mai puţin semnificativ

TABELUL 2.3. **Transferuri pe 16 biți (continuare)**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect: Binar	Codul obiect: Hexa	Indicatorii de condiții: S Z H P/V N C
LD (nn),HL	SHLD nn	(nn+1) ← H (nn) ← L	00 100 010 ← n → ← n →	22	. . × . × . .
LD (nn),dd		(nn+1) ← dd_H (nn) ← dd_L	11 101 101 01 dd0 011 ← n → ← n →	ED	. × . × .
LD (nn),IX		(nn+1) ← IX_H (nn) ← IX_L	11 011 101 00 100 010 ← n → ← n →	DD 22	. × . × .
LD (nn),IY		(nn+1) ← IY_H (nn) ← IY_L	11 111 101 00 100 010 ← n → ← n →	FD 22	. × . × .
LD SP,HL	SPHL	SP ← HL	11 111 001	F9	. . × . × . . .
LD SP,IX		SP ← IX	11 011 101 11 111 001	DD F9	. × . × .
LD SP,IY		SP ← IY	11 111 101 11 111 001	FD F9	. . × . × .
PUSH qq	PUSH rp PUSH PSW	(SP−2) ← qq_L (SP−1) ← qq_H SP ← SP−2	11 qq0 101		. × . × .
PUSH IX		(SP−2) ← IX_L (SP−1) ← IX_H SP ← SP−2	11 011 101 11 100 101	DD E5	. × . × .
PUSH IY		(SP−2) ← IY_L (SP−1) ← IY_H SP ← SP−2	11 111 101 11 100 101	FD E5	. × . × .
POP qq	POP rp POP PSW	qq_H ← (SP+1) qq_L ← (SP) SP ← SP+2	11 qq0 001		. × . × .
POP IX		IX_H ← (SP+1) IX_L ← (SP) SP ← SP+2	11 011 101 11 100 001	DD E1	. × . × .
POP IY		IY_H ← (SP+1) IY_L ← (SP) SP ← SP+2	11 111 101 11 100 001	FD E1	. × . × . . .

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)	COL(3)	COH(3)	SML(3)	SMH(3)	
ECO(4)+ ECO(4)	COL(3)	COH(3)	SML(3)	SMH(3)	
ECO(4)+ ECO(4)	COL(3)	COH(3)	SML(3)	SMH(3)	
ECO(4)+ ECO(4)	COL(3)	COH(3)	SML(3)	SMH(3)	
ECO(6					
ECO(4)+ ECO(6)					
ECO(4)+ ECO(6)					
ECO(5) SP−1 →	SSH(3) SP−1 →	SSL(3)			
ECO(4)+ ECO(5) SP−1 →	SSH(3) SP−1 →	SSL(3)			
ECO(4)+ ECO(5) SP−1 →	SSH(3) SP−1 →	SSL(3)			
ECO(4)	CSL(3) SP+1 →	CSH(3) SP+1 →			
ECO(4)+ ECO(4)	CSL(3) SP+1 →	CSH(3) SP+1 →			
ECO(4)+ ECO(4)	CSL(3) SP+1 →	CSH(3) SP+1 →			

TABELUL 2.4. **Schimburi între registre, transfer de blocuri, căutări**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiții
			Binar	Hexa	S Z H P/V N C
EX DE,HL	XCHG	DE ↔ HL	11 101 011	EB	. . × . × . . .
EX AF,AF'		AF ↔ AF'	00 001 000	08	. . × . × . . .
EXX		BC ↔ BC' DE ↔ DE' HL ↔ HL'	11 011 001	D9	. . × . × . . .
EX (SP),HL	XTHL	H ↔ (SP+1) L ← (SP)	11 100 011	E3	. . × . × . . .
EX (SP),IX		IX_H ↔ (SP+1) IX_L ↔ (SP)	11 011 101 11 100 011	DD E3	. . × . × . . .
EX (SP),IY		IY_H ↔ (SP+1) IY_L ↔ (SP)	11 111 101 11 100 011	FD E3	. . × . × . . .
LDI		(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC−1	11 101 101 10 100 000	ED A0	. . × 0 × ● 0 .
LDIR		(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC−1 se repetă până când BC=0	11 101 101 10 110 000	ED B0	. . × 0 × 0 0 .
LDD		(DE) ← (HL) DE ← DE−1 HL ← HL−1 BC ← BC−1	11 101 101 10 101 000	ED A8	. . × 0 × ● 0 .
LDDR		(DE) ← (HL) DE ← DE−1 HL ← HL+1 BC ← BC−1 se repetă până când BC=0	11 101 101 10 111 000	ED B8	. . × 0 × 0 0 .
CPI		A − (HL) HL ← HL+1 BC ← BC−1	11 101 101 10 100 001	ED A1	● ● × ● × ● 1 .
CPIR		A − (HL) HL ← HL+1 BC ← BC−1 se repetă până când A=(HL) sau BC=0	11 101 101 10 110 001	ED B1	● ● × ● × ● 1 .

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)					
ECO(4)					
ECO(4)					
ECO(4)	CSL(3) [SP+1]	CSH(4)	SSH(3) [SP−1]	SSL(5)	
ECO(4)+ ECO(4)	CSL(3) [SP+1]	CSH(4)	SSH(3) [SP−1]	SSL(5)	
ECO(4)+ ECO(4)	CSL(3) [SP+1]	CSH(4)	SSH(3) [SP−1]	SSL(5)	
ECO(4)+ ECO(4)	CM(3)	SM(5)			Nota 3 Nota 4
ECO(4)+ ECO(4)	CM(3)	SM(5)	OI(5) numai când BC≠0		
ECO(4)+ ECO(4)	CM(3)	SM(5)			Nota 4
ECO(4)+ ECO(4)	CM(3)	SM(5)	OI(5) numai când BC≠0		
ECO(4)+ ECO(4)	CM(3)	OI(5)			Nota 4 Nota 5 Nota 6
ECO(4)+ ECO(4)	CM(3)	OI(5)	OI(5) numai când BC≠0		Nota 4 Nota 6

TABELUL 2.4. Schimburi între registre, transfer de blocuri, căutări (continuare)

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect Binar	Codul obiect Hexa	Indicatorii de condiții S Z H P/V N C
CPD		A − (HL) HL ← HL−1 BC ← BC−1	11 101 101 10 101 001	ED A9	• • × • × • 1 .
CPDR		A − (HL) HL ← HL−1 BC ← BC−1 se repetă până când A=(HL) sau BC=0	11 101 101 10 111 001	ED B9	• • × • × • 1 .

TABELUL 2.5. Operații aritmetice și logice pe 8 biți

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect Binar	Codul obiect Hexa	Indicatorii de condiții S Z H P/V N C
ADD A,r	ADD r	A ← A + r	10<000> r		• • × • × V 0 •
ADD A,n	ADI n	A ← A + n	11<000>110 ← n →		• • × • × V 0 •
ADD A,(HL)	ADD M	A ← A + (HL)	10<000>110		• • × • × V 0 •
ADD A,(IX+d)		A ← A + (IX+d)	11 011 101 10<000>110 ← d →	DD	• • × • × V 0 •
ADD A,(IY+d)		A ← A + (IY+d)	11 111 101 10<000>110 ← d →	FD	• • × • × V 0 •
ADC A,s	ADC r ADC M ACI n	A ← A + s + CY	<001>		• • × • × V 0 •
SUB s	SUB r SUB M SUI n	A ← A − s − CY	<010>		• • × • × V 0 •
SBC A,s	SBB r SBB M SBI n	A ← A − s − CY	<011>		• • × • × V 0 •
AND s	ANA r ANA M ANI n	A ← A ∧ s	<100>		• • × 1 × P 0 0

Ciclii-mașină					Observații și note
M_1	M_2	M_3	M_4	M_5	
ECO(4)+ ECO(4)	CM(3)	OI(5)			Nota 4 Nota 6
ECO(4)+ ECO(4)	CM(3)	OI(5)	OI(5) numai când BC≠0		Nota 4 Nota 6

Ciclii-mașină					Observații și note
M_1	M_2	M_3	M_4	M_5	
ECO(4)					r — Registru 000 — B 001 — C 010 — D 011 — E 100 — H 101 — L 111 — A
ECO(4)	CO(3)				
ECO(4)	CM(3)				
ECO(4)+ ECO(4)	CO(3)	OI(5)	CM(3)		
ECO(4)+ ECO(4)	CO(3)	OI(5)	CM(3)		
La fel ca la ADD, funcție de s					s poate fi r, n, (HL), (IX+d) sau (IY+d), ca în cazul instrucțiunilor ADD. Pentru a se obține codul-obiect se înlocuiesc biții <000> din codul-obiect al instrucțiunii ADD corespunzătoare cu biții indicați.
La fel ca la ADD, funcție de s					
La fel ca la ADD, funcție de s					
La fel ca la ADD, funcție de s					

TABELUL 2.5. **Operaţii aritmetice şi logice pe 8 biţi (continuare)**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiţii							
			Binar	Hexa	S	Z		H		P/V	N	C
OR s	ORA r ORA M ORI n	A ← A ∨ s	<110>		•	•	×	0	×	P	0	0
XOR s	XRA r XRA M XRI n	A ← A ⊕ s	<101>		•	•	×	0	×	P	0	0
CP s	CMP r CMP M CPI n	A − s	<111>		•	•	×	•	×	V	1	•
INC r	INR r	r ← r + 1	00 r <100>		•	•	×	•	×	V	0	.
INC (HL)	INR M	(HL) ← (HL) + 1	00 110 <100>		•	•	×	•	×	V	0	.
INC (IX+d)		(IX+d) ← (IX+d) + 1	11 011 101 00 110 <100> ← d →	DD	•	•	×	•	×	V	0	.
INC (IY+d)		(IY+d) ← (IY+d) + 1	11 111 101 00 110 <100> ← d →	FD	•	•	×	•	×	V	0	.
DEC m	DCR r DCR M	m ← m − 1	<101>		•	•	×	•	×	V	1	.

TABELUL 2.6. **Operaţii aritmetice generale şi operaţii de comandă a UC-Z80**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiţii							
			Binar	Hexa	S	Z		H		P/V	N	C
DAA	DAA	Converteşte conţinutul acumulatorului într-un format BCD după adunări sau scăderi cu operanzii în format BCD	00 100 111	27	•	•	×	•	×	P	.	•
CPL	CMA	A ← $\overline{A}$	00 101 111	2F	.	.	×	1	×	.	1	.
NEG		A ← 0 − A	11 101 101 01 000 100	ED 44	•	•	×	•	×	V	1	•

Ciclii-mașină					Observații și note
M_1	M_2	M_3	M_4	M_5	
La fel ca la ADD, funcție de s					
La fel ca la ADD, funcție de s					
La fel ca la ADD, funcție de s					
ECO(4)					
ECO(4)	CM(4)	SM(3)			
ECO(4)+ ECO(4)	CO(3)	OI(5)	CM(4)	SM(3)	
ECO(4)+ ECO(4)	CO(3)	OI(5)	CM(4)	SM(3)	
La fel ca la INC, funcție de m					m poate fi r, (HL), (IX+d) sau (IY+d); ca în cazul instrucțiunilor INC. Pentru a se afla codul-obiect se înlocuiesc biții <100> cu <101>

Ciclii-mașină					Observații și note
M_1	M_2	M_3	M_4	M_5	
ECO(4)					Nota 7
ECO(4)					Complement față de 1. Nota 8
ECO(4)+ ECO(4)					Complement față de 2. Nota 9

TABELUL 2.6. **Operaţii aritmetice generale şi operaţii de comandă a UC-Z80 (continuare)**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiţii							
			Binar	Hexa	S	Z		H		P/V	N	C
CCF	CMC	CY ← $\overline{CY}$	00 111 111	3F	.	.	×	×	×	.	0	●
SCF	STC	CY ← 1	00 110 111	37	.	.	×	0	×	.	0	1
NOP	NOP		00 000 000	00	.	.	×	.	×	.	.	.
HALT	HLT	Oprire UC-Z80	01 110 110	76	.	.	×	.	×	.	.	.
DI	DI	IFF1,2 ← 0	11 110 011	F3	.	.	×	.	×	.	.	.
EI	EI	IFF1,2 ← 1	11 111 011	FB	.	.	×	.	×	.	.	.
IM 0		Stabilire mod întreruperi 0	11 101 101 01 000 110	ED 46	.	.	×	.	×	.	.	.
IM 1		Stabilire mod întreruperi 1	11 101 101 01 010 110	ED 56	.	.	×	.	×	.	.	.
IM 2		Stabilire mod întreruperi 2	11 101 101 01 011 110	ED 5E	.	.	×	.	×	.	.	.

TABELUL 2.7. **Operaţii aritmetice pe 16 biţi**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiţii							
			Binar	Hexa	S	Z		H		P/V	N	C
ADD HL,ss	DAD rp	HL ← HL + ss	00 ss1 001		.	.	×	×	×	.	0	●
ADC HL,ss		HL ← HL + ss + CY	11 101 101 01 ss1 010	ED	●	●	×	×	×	V	0	●
SBC HL,ss		HL ← HL − ss − CY	11 101 101 01 ss0 010	ED	●	●	×	×	×	V	1	●
ADD IX,pp		IX ← IX + pp	11 011 101 01 pp1 001	DD	.	.	×	×	×	.	0	●
ADD IY,rr		IY ← IY + rr	11 111 101 00 rr1 001	FD	.	.	×	×	×	.	0	●
INC ss	INX rp	ss ← ss + 1	00 ss0 011		.	.	×	.	×	.	.	.
INC IX		IX ← IX + 1	11 011 101 00 100 011	DD 23	.	.	×	.	×	.	.	.
INC IY		IY ← IY + 1	11 111 101 00 100 011	FD 23	.	.	×	.	×	.	.	.

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)					Întreruperile nu sunt eşantionate la sfârşitul instrucţiunilor EI şi DI
ECO(4)					
ECO(4)					
ECO(4)					
ECO(4)					
ECO(4)					
ECO(4)+ ECO(4)					
ECO(4)+ ECO(4)					
ECO(4)+ ECO(4)					

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)	OI(4)	OI(3)			
ECO(4)+ ECO(4)	OI(4)	OI(3)			
ECO(4)+ ECO(4)	OI(4)	OI(3)			
ECO(4)+ ECO(4)	OI(4)	OI(3)			
ECO(4)+ ECO(4)	OI(4)	OI(3)			
ECO(6)					
ECO(4)+ ECO(6)					
ECO(4)+ ECO(6)					

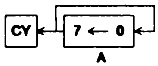
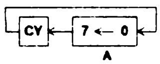
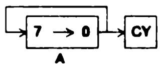
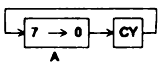
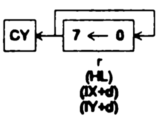
ss	Registre
00	BC
01	DE
10	HL
11	SP

pp	Registre
00	BC
01	DE
10	IX
11	SP

TABELUL 2.7. **Operații aritmetice pe 16 biți (continuare)**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiții
			Binar	Hexa	S Z H P/V N C
DEC ss	DCX rp	ss ← ss − 1	00 ss1 011		. . × . × . . .
DEC IX		IX ← IX − 1	11 011 101 00 101 011	DD 2B	. . × . × . . .
DEC IY		IY ← IY − 1	11 111 101 00 101 011	FD 2B	. . × . × . . .

TABELUL 2.8. **Rotații și deplasări**

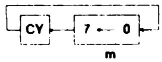
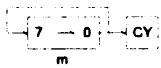
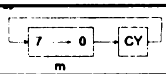
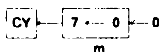
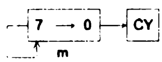
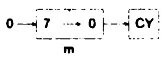
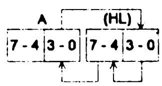
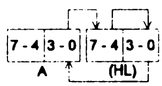
Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiții
			Binar	Hexa	S Z H P/V N C
RLCA	RLC		00 000 111	07	. . × 0 × . 0 ♦
RLA	RAL		00 010 111	17	. × 0 × . 0 ♦
RRCA	RRC		00 001 111	0F	. . × 0 × . 0 ♦
RRA	RAR		00 011 111	1F	. . × 0 × . 0 ♦
RLC r			11 001 011 00<000> r	CB	♦ ♦ × 0 × P 0 ♦
RLC (HL)			11 001 011 00<000>110	CB	♦ ♦ × 0 × P 0 ♦
RLC (IX+d)			11 011 101 11 001 011 ← d → 00<000>110	DD CB	♦ ♦ × 0 × P 0 ♦

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(6)					
ECO(4)+ ECO(6)					
ECO(4)+ ECO(6)					

rr	Registre
00	BC
01	DE
10	IY
11	SP

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)					
ECO(4)					
ECO(4)					
ECO(4)					
ECO(4)					
ECO(4)+ ECO(4)					
ECO(4)+ ECO(4)					
ECO(4)+ ECO(4)	CO(3)	OI(5)	CM(4)	SM(3)	

TABELUL 2.8. **Rotaţii şi deplasări (continuare)**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect: Binar	Codul obiect: Hexa	Indicatorii de condiţii: S Z H P/V N C
RLC (IY+d)			11 111 101 11 001 011 ← d → 00<000>110	FD CB	↕ ↕ × 0 × P 0 ↕
RL m			<010>		↕ ↕ × 0 × P 0 ↕
RRC m			<001>		↕ ↕ × 0 × P 0 ↕
RR m			<011>		↕ ↕ × 0 × P 0 ↕
SLA m			<100>		↕ ↕ × 0 × P 0 ↕
SRA m			<101>		↕ ↕ × 0 × P 0 ↕
SRL m			<111>		↕ ↕ × 0 × P 0 ↕
RLD			11 101 101 01 101 111	ED 6F	↕ ↕ × 0 × P 0 .
RRD			11 101 101 01 100 111	ED 67	↕ ↕ × 0 × P 0 .

TABELUL 2.9. **Prelucrări pe bit**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect: Binar	Codul obiect: Hexa	Indicatorii de condiţii: S Z H P/V N C
BIT b,r		$Z \leftarrow \overline{r}_b$	11 001 011 01 b r	CB	× ↕ × 1 × × 0 .
BIT b,(HL)		$Z \leftarrow \overline{(HL)}_b$	11 001 011 01 b 110	CB	× ↕ × 1 × × 0 .
BIT b,(IX+d)		$Z \leftarrow \overline{(IX+d)}_b$	11 011 101 11 001 011 ← d → 01 b 110	DD CB	× ↕ × 1 × × 0 .

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)+ ECO(4)	CO(3)	OI(5)	CM(4)	SM(3)	
La fel ca RLC, funcţie de m					m=r, (HL), (IX+d), (IY+d) Pentru a obţine codul-obiect se înlocuiesc biţii <000> din codul-obiect al instrucţiunii RLC corespunzătoare cu biţii indicaţi
La fel ca RLC, funcţie de m					
La fel ca RLC, funcţie de m					
La fel ca RLC, funcţie de m					
La fel ca RLC, funcţie de m					
La fel ca RLC, funcţie de m					
ECO(4)+ ECO(4)	CM(3)	OI(4)	SM(3)		
ECO(4)+ ECO(4)	CM(3)	OI(4)	SM(3)		

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)+ ECO(4)					r — Registru 000 — B 001 — C 010 — D 011 — E 100 — H 101 — L 111 — A
ECO(4)+ ECO(4)	CM(4)				
ECO(4)+ ECO(4)	CO(3)	OI(5)	CM(4)		

TABELUL 2.9. **Prelucrări pe bit (continuare)**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiţii
			Binar	Hexa	S Z H P/V N C
BIT b,(IY+d)		$Z \leftarrow \overline{(IY+d)_b}$	11 111 101 11 001 011 ← d → 01 b 110	FD CB	× ‡ × 1 × × 0 .
SET b,r		$r_b \leftarrow 1$	11 001 011 <11> b r	CB	. × . × . . .
SET b,(HL)		$(HL)_b \leftarrow 1$	11 001 011 <11> b 110	CB	. × . × . .
SET b,(IX+d)		$(IX+d)_b \leftarrow 1$	11 011 101 11 001 011 ← d → <11> b 110	DD CB	. . × . × . . .
SET b,(IY+d)		$(IY+d)_b \leftarrow 1$	11 111 101 11 001 011 ← d → <11> b 110	FD CB	. × . × . . .
RES b,m		$m_b \leftarrow 0$	<10>	FD	. × . × . . .

TABELUL 2.10. **Salturi**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiţii
			Binar	Hexa	S Z H P/V N C
JP nn	**JMP nn**	PC ← nn	11 000 011 ← n → ← n →	C3	. × . × .
JP cc,nn	**Jcc nn**	Dacă cc adevărat PC ← nn, altfel continuă	11 cc 010 ← n → ← n →		. × . × . . .
JR e		PC ← PC + e	00 011 000 ← e−2 →	18	. × . × . . .
JR C,e		Dacă C=0 continuă Dacă C=1 PC ← PC+e	00 111 000 ← e−2 →	38	. × . × . . .

Ciclii-mașină					Observații și note
M_1	M_2	M_3	M_4	M_5	
ECO(4)+ ECO(4)	CO(3)	OI(5)	CM(4)		b | Bit testat 000 | 0 001 | 1 010 | 2 011 | 3 100 | 4 101 | 5 110 | 6 111 | 7
ECO(4)+ ECO(4)					
ECO(4)+ ECO(4)	CM(4)	SM(3)			
ECO(4)+ ECO(4)	CO(3)	OI(5)	CM(4)	SM(3)	
ECO(4)+ ECO(4)	CO(3)	OI(5)	CM(4)	SM(3)	
La fel ca SET, funcție de m					m=r, (HL), (IX+d), (IY+d) Pentru a obține codul-obiect se înlocuiesc biții < 11 > din codul-obiect al instrucțiunii SET corespunzătoare cu biții < 10 >.

Ciclii-mașină					Observații și note
M_1	M_2	M_3	M_4	M_5	
ECO(4)	COL(3)	COH(3)			cc | Condiția 000 | NZ (non-zero) 001 | Z (zero) 010 | NC (non-carry) 011 | C (carry) 100 | PO (impar) 101 | PE (par) 110 | PE (pozitiv) 111 | M (negativ)
ECO(4)	COL(3)	COH(3)			
ECO(4)	CO(3)	OI(5)			
ECO(4)	CO(3)	OI(5)*			

TABELUL 2.10. **Salturi (continuare)**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiţii
			Binar	Hexa	S Z H P/V N C
JR NC,e		Dacă C=1 continuă Dacă C=0 PC ← PC+e	00 110 000 ← e−2 →	30	. . × . × . .
JR Z,e		Dacă Z=0 continuă Dacă Z=1 PC ← PC+e	00 101 000 ← e−2 →	28	. × . × . . .
JR NZ,e		Dacă Z=1 continuă Dacă Z=0 PC ← PC+e	00 100 000 ← e−2 →	20	. × . × . .
JP (HL)	PCHL	PC ← HL	11 101 001	E9	. . × . × . . .
JP (IX)		PC ← IX	11 011 101 11 101 001	DD E9	. . × . × . . .
JP (IY)		PC ← IY	11 111 101 11 101 001	FD E9	. × . × . . .
DJNZ e		B ← B − 1 Dacă B=0 continuă Dacă B≠0 PC ← PC+e	00 010 000 ← e−2 →	10	. . × . × .

TABELUL 2.11. **Apeluri de subrutine, reveniri, instrucţiuni de restart**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiţii
			Binar	Hexa	S Z H P/V N C
CALL nn	CALL nn	(SP−1) ← PC_H (SP−2) ← PC_L PC ← nn	11 001 101 ← n → ← n →	CD	. . × . × . .
CALL cc,nn	Ccc nn	Dacă cc adevărat (=1) CALL nn, altfel continuă	11 cc 100 ← n → ← n →		. × . × .
RET	RET	PC_L ← (SP) PC_H ← (SP+1)	11 001 001	C9	. × . × . . .
RET cc	Rcc	Dacă cc=0 continuă Dacă cc=1 RET	11 cc 000		. . × . × . . .

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)	CO(3)	OI(5)*			* M3 numai dacă se îndeplineşte condiţia (la salt)
ECO(4)	CO(3)	OI(5)*			
ECO(4)	CO(3)	OI(5)*			
ECO(4)					e reprezintă un număr scris în complement faţă de 2 cuprins între −126 şi +129
ECO(4)+ ECO(4)					
ECO(4)+ ECO(4)					
ECO(4)	CO(3)	OI(5) dacă B≠0			

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)	COL(3)	COH(4) SP−1 →	SSH(3) SP−1 →	SSL(3)	
ECO(4) ECO(4)	COL(3) COL(3)	COH(3) COH(4) SP−1 →	 SSH(3) SP−1 →	 SSL(3)	cc=0 cc=1
ECO(4)	CSL(3) SP+1 →	CSH(3) SP+1 →			
ECO(5)	CSL(3) dacă cc=1 SP+1 →	CSH(3) dacă cc=1 SP+1 →			

cc	Condiţia
000	NZ (non-zero)
001	Z (zero)
010	NC (non-carry)
011	C (carry)
100	PO (impar)
101	PE (par)
110	P (pozitiv)
111	M (negativ)

TABELUL 2.11. **Apeluri de subrutine, reveniri, instrucţiuni de restart (continuare)**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiţii
			Binar	Hexa	S Z H P/V N C
RETI		Revenire din întrerupere	11 101 101 01 001 101	ED 4D	. × . × . . .
RETN		Revenire din întrerupere nemascabilă	11 101 101 01 000 101	ED 45	. . × . × . . .
RST p	RST t	(SP−1) ← PC_H (SP−2) ← PC_L PC_H ← 0 PC_L ← p	11 t 111		. × . × . . .

TABELUL 2.12. **Operaţii de intrare/ieşire**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiţii
			Binar	Hexa	S Z H P/V N C
IN A,(n)	IN n	A ← (n)	11 011 011 ← n →	DB	. . × . × . . .
IN r,(C)		r ← (C)	11 101 101 01 r 000	ED	↕ ↕ × ↕ × P 0 .
INI		(HL) ← (C) B ← B − 1 HL ← HL + 1	11 101 101 10 100 010	ED A2	× ↕ × × × × ↕ ×
INIR		(HL) ← (C) B ← B − 1 HL ← HL + 1 se repetă până când B=0	11 101 101 10 110 010	ED B2	× 1 × × × × ↕ ×
IND		(HL) ← (C) B ← B − 1 HL ← HL − 1	11 101 101 10 101 010	ED AA	× ↕ × × × × ↕ ×
INDR		(HL) ← (C) B ← B − 1 HL ← HL − 1 se repetă până când B=0	11 101 101 10 111 010	ED BA	× 1 × × × × ↕ ×
OUT (n),A	OUT n	(n) ← A	11 010 011 ← n →	D3	. × . × . ↕ ×

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)+ ECO(4)	CSL(3) [SP+1]	CSH(3) [SP+1]			t / p: 000 – 00H; 001 – 08H; 010 – 10H; 011 – 18H; 100 – 20H; 101 – 28H; 110 – 30H; 111 – 38H
ECO(4)+ ECO(4)	CSL(3) [SP+1]	CSH(3) [SP+1]			
ECO(5) [SP−1]	SSH(3) [SP−1]	SSL(3)			

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)	CO(3)	CPO(4)			$A_0 \div A_7 \leftarrow n$, $A_8 \div A_{15} \leftarrow A$
ECO(4)+ ECO(4)	CPO(4)				$A_0 \div A_7 \leftarrow C$, $A_8 \div A_{15} \leftarrow B$ Nota 7
ECO(4)+ ECO(5)	CPO(4)	SM(3)			$A_0 \div A_7 \leftarrow C$, $A_8 \div A_{15} \leftarrow B$ Nota 8
ECO(4)+ ECO(5)	CPO(4)	SM(3)	OI(5) dacă $B \neq 0$		$A_0 \div A_7 \leftarrow C$, $A_8 \div A_{15} \leftarrow B$
ECO(4)+ ECO(5)	CPO(4)	SM(3)			$A_0 \div A_7 \leftarrow C$, $A_8 \div A_{15} \leftarrow B$ Nota 8
ECO(4)+ ECO(5)	CPO(4)	SM(3)	OI(5) dacă $B \neq 0$		$A_0 \div A_7 \leftarrow C$, $A_8 \div A_{15} \leftarrow B$
ECO(4)	CO(3)	SPO(4)			$A_0 \div A_7 \leftarrow n$, $A_8 \div A_{15} \leftarrow A$

TABELUL 2.12. **Operaţii de intrare/ieşire (continuare)**

Mnemonica Z80	Mnemonica 8080	Descrierea simbolică	Codul obiect		Indicatorii de condiţii
			Binar	Hexa	S Z H P/V N C
OUT (C),r		(C) ← r	11 101 101 01 r 001	ED	. × . × . • ×
OUTI		(C) ← (HL) B ← B − 1 HL ← HL + 1	11 101 101 10 100 011	ED A3	× • × × × × • ×
OTIR		(C) ← (HL) B ← B − 1 HL ← HL + 1 se repetă până când B=0	11 101 101 10 110 011	ED B3	× 1 × × × × • ×
OUTD		(C) ← (HL) B ← B − 1 HL ← HL − 1	11 101 101 10 101 011	ED AB	× • × × × × • ×
OTDR		(C) ← (HL) B ← B − 1 HL ← HL − 1 se repetă până când B=0	11 101 101 10 111 011	ED BB	× 1 × × × × • ×

NOTE:

1. Exemplu de transfer într-o locaţie de memorie folosind alocare indexată pentru destinaţie şi adresare imediată pentru sursă:

```
LD(IX-31),44H
```

Instrucţiunea va apărea în memorie scrisă în următoarea ordine:

Adresa A	DD	} Cod-operaţie
A+1	36	
A+2	E1	} Deplasament în complement faţă de 2
A+3	44	} Operand

2. Exemplu de instrucţiune de transfer pe 16 biţi utilizând adresarea imediată pentru sursă şi adresarea de registru pentru destinaţie:

```
LD BC, 176AH
```

Instrucţiunea va apărea în memorie scrisă astfel:

Ciclii-maşină					Observaţii şi note
M_1	M_2	M_3	M_4	M_5	
ECO(4)+ ECO(4)	SPO(4)				$A_0 \div A_7 \leftarrow C$, $A_8 \div A_{15} \leftarrow B$ Nota 7
ECO(4)+ ECO(5)	CM(3)	SPO(4)			$A_0 \div A_7 \leftarrow C$, $A_8 \div A_{15} \leftarrow B$ Nota 8
ECO(4)+ ECO(5)	CM(3)	SPO(4)	OI(5) dacă $B \neq 0$		$A_0 \div A_7 \leftarrow C$, $A_8 \div A_{15} \leftarrow B$
ECO(4)+ ECO(5)	CM(3)	SPO(4)	OI(5)		$A_0 \div A_7 \leftarrow C$, $A_8 \div A_{15} \leftarrow B$ Nota 8
ECO(4)+ ECO(5)	CM(3)	SPO(4)	OI(5) dacă $B \neq 0$		$A_0 \div A_7 \leftarrow C$, $A_8 \div A_{15} \leftarrow B$

Adresa A	01	} Cod-operaţie
	6A	} Operand mai puţin semnificativ → C
	17	} Operand mai semnificativ → B

3. Instrucţiunile pentru transfer de blocuri de date, LDI, LDIR, LDD şi LDDR, utilizează toate, în acelaşi mod, următoarele registre:

HL, pentru a adresa locaţia-sursă,
DE, pentru a adresa locaţia-destinaţie şi
BC, ca numărător de octeţi.

Pentru folosirea corectă a celor patru instrucţiuni de transfer trebuie ca registrele HL, DE şi BC să fie iniţializate, în prealabil, conform operaţiei ce se doreşte a fi executată.

LDI, *Load and Increment*, încărcare şi incrementare, transferă un octet din locaţia adresată de HL în locaţia adresată cu DE. Registrele HL şi DE se incrementează pentru a fi pregătite să adreseze următoarele locaţii, iar numărătorul de octeţi BC se decrementează. Instrucţiunea este utilă atunci când între transferul a doi octeţi sunt necesare prelucrări suplimentare. Dacă aceste prelucrări nu sunt necesare se poate folosi instrucţiunea LDIR, *Load, Increment and Repeat*, încărcare, incrementare şi repetare, care repetă transferul până când numărătorul de octeţi devine zero. Menţionăm că lungimea maximă a unui bloc poate fi de 64 Kocteţi şi că zonele de transfer se pot suprapune.

Instrucţiunile LDD şi LDDR sunt asemănătoare cu LDI şi LDIR, diferenţa constând numai în aceea că registrele de adresare, HL şi DE, sunt *decrementate* după fiecare transfer. Deci LDI şi LDIR iniţiază transferul cu adresa de început a blocului, în timp ce LDD şi LDDR cu adresa de sfârşit.

Exemplu:

```
HL     : 3333H                               HL     : 3334H
(3333H):   55H   Executând instrucţiunea LDI  (3333H):   55H
DE     : 7777H   se obţine următoarea         DE     : 7778H
(7777H):   99H   situaţie:                    (7777H):   55H
BC     :   0BH                               BC     :   0AH
```

4. Indicatorul P/V se pune pe "0" dacă BC−1=0. În celelalte cazuri P/V rămâne "1".

5. Instrucţiunile de căutare, CPI, CPIR, CPD, şi CPDR, compară acumulatorul cu conţinutul locaţiei adresate de HL. Rezultatul comparaţiei se memorează în indicatorul de condiţii Z. Ca şi în cazul transferurilor, CPI, *Compare and Increment*, după ce compară acumulatorul cu (HL), incrementează registrul HL, şi decrementează numărătorul de octeţi. Dacă nu sunt necesare prelucrări suplimentare între două comparări de octeţi succesivi, se poate utiliza instrucţiunea CPIR, *Compare, Increment and Repeat*, care repetă comparaţia fie până când A=(HL), fie până când BC=0.

CPD şi CPDR sunt asemănătoare cu CPI şi CPIR diferenţa constând numai în faptul că registrul HL este *decrementat* după fiecare operaţie.

Exemplu:

```
HL     : 1111H                                 HL     : 1114H
A      :  0F3H                                 A      :  0F3H
BC     : 0007H                                 BC     : 0004H
(1111H):   77H   Executând instrucţiunea CPIR,  (1111H):   77H
(1112H):   01H   se obţine următoarea           (1112H):   01H
(1113H):  0F3H   situaţie:                      (1113H):  0F3H
P/V    :     0                                 P/V    :     1
Z      :     0                                 Z      :     1
```

6. Indicatorul Z se pune pe "1" dacă A=(HL). În celelalte situaţii Z=0.

7. Executând o operaţie de adunare a două numere, de exemplu, 17(BCD) şi 39(BCD), se obţine simplu, în aritmetică zecimală, rezultatul 56. Lucrând cu reprezentări binare:

```
 0001 0111
+0011 1001
----------
 0101 0000 = 50
```

rezultatul este incorect. Instrucţiunea DAA ajustează acest rezultat pentru a obţine reprezentarea corectă BCD a sumei:

```
 0101 0000
+0000 0110
----------
 0101 0110 = 56
```

8. Dacă acumulatorul conţine 0101 1101, după execuţia instrucţiunii CPI va conţine 1010 0010.

9. Acumulatorul se modifică după execuţia instrucţiunii NEG din 0101 1101 în 1010 0011.

10. Valorile lui *r* sunt aceleaşi ca în tabelul 2.9; dacă *r*=110, se modifică numai indicatorii de condiţii.

11. Dacă rezultatul scăderii B−1 este "0", indicatorul Z se poziţionează pe "1"; în celelalte situaţii Z=0.

2.2. CIRCUITUL PENTRU COMANDA INTRĂRILOR/IEŞIRILOR PARALELE PIO-Z80

Circuitul pentru comanda intrărilor/ieşirilor paralele, PIO-Z80, este destinat conectării echipamentelor periferice cu interfeţe paralele la unităţi centrale realizate cu microprocesoare Z80.

PIO-Z80 cuplează perifericele prin intermediul a două *port*-uri de I/E de 8 biţi concepute să lucreze independent, *port*-ul A şi *port*-ul B. Fiecare *port* are asociate două semnale, READY şi $\overline{\text{STROBE}}$, cu ajutorul cărora poate comanda transferul datelor. Ieşirea READY indică perifericului faptul că *port*-ul este pregătit, gata, pentru un transfer de date. Intrarea $\overline{\text{STROBE}}$ de la periferic semnalează când a apărut un transfer.

Circuitul PIO-Z80 poate fi programat să lucreze în unul din următoarele 4 moduri:

Modul 0, ieşire-octet
Modul 1, intrare-octet
Modul 2, intrare/ieşire-octet
Modul 3, intrare/ieşire pe bit

Ambele *port*-uri de I/E se pot programa să lucreze în *modul 0*. Cele două *port*-uri sunt prevăzute cu registre de ieşire adresabile direct de către UC, datele putând fi transmise către oricare din ele în orice moment. După ce datele au fost scrise într-un *port*, se va acţiona ieşirea READY corespunzătoare, indicându-se în acest fel perifericului asociat că informaţia este disponibilă şi se poate efectua transferul. După transferarea datelor, dispozitivul extern va răspunde activând intrarea $\overline{\text{STROBE}}$, care, la rândul ei, va putea conduce la generarea unei întreruperi, în cazul în care aceasta a fost în prealabil validată.

În *modul 1* cele două *port*-uri ale circuitului PIO-Z80 pot fi utilizate pentru a transfera informaţia de la periferic spre UC. Fiecare din *port*-uri are un registru de intrare adresabil direct de unitatea centrală. Când urmează să citească un octet din *port*, PIO poziţionează întâi semnalul READY. Perifericul detectează acest lucru şi, ca răspuns, plasează datele pe liniile de intrare/ieşire, eşantionându-le cu ajutorul semnalului $\overline{\text{STROBE}}$. Semnalul $\overline{\text{STROBE}}$ e utilizat de PIO pentru a memora datele în registrul de intrare şi a declanşa o cerere de întrerupere, dacă aceasta a fost în prealabil validată.

În *modul 2*, bidirecţional pe octet, se foloseşte pentru transfer un singur *port*, A, împreună cu semnalele de comandă ale ambelor *port*-uri. Celălalt *port*, B, trebuie programat în modul 3 şi mascat. O operaţie de ieşire este similară cu un transfer în modul 0, cu observaţia că datele se pot genera numai când linia $\overline{\text{STROBE}}$ a *port*-ului A e pe "0". În intrare funcţionarea este asemănătoare cu lucrul în modul 1 cu menţiunea că pentru dialog se întrebuinţează semnalele de comandă şi întreruperea asociate *port*-ului B.

Modul 3, utilizabil de ambele *port*-uri, permite definirea individuală a biţilor ca intrări sau ieşiri. Nu se folosesc semnalele de comandă, întreruperea generându-se la activarea unei singure intrări sau a tuturor. Prin programare se specifică nivelul activ al fiecărei intrări, "0" sau "1", şi condiţia logică ce va

trebui să fie îndeplinită pentru a se genera întreruperea: SAU - o singură intrare activă, ŞI - toate intrările active. De exemplu dacă unul din *port*-uri este programat să aibă intrări active pe "0" şi condiţia logică este ŞI atunci circuitul PIO-Z80 va genera o cerere de întrerupere când toate intrările acelui *port* trec pe "0".

Precizăm că numerotarea modurilor are şi o semnificaţie mnemotehnică: 0=*O*ut (ieşire), 1=*I*n (intrare), 2=*Bi*direcţional.

2.2.1. STRUCTURA CIRCUITULUI PIO-Z80

În figura 2.19 este dată schema bloc a unui circuit PIO-Z80. După cum se observă, el este structurat în jurul unei magistrale interne la care sunt conectate logica de comandă internă, partea de interfaţă cu UC-Z80, cele două *port*-uri de I/E şi logica de comandă a întreruperii.

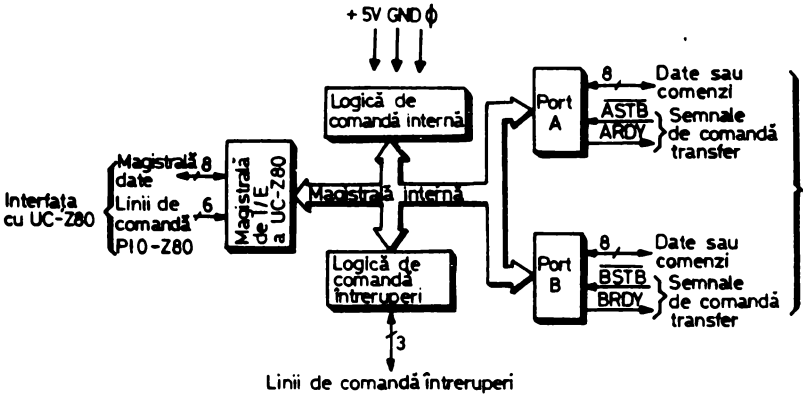
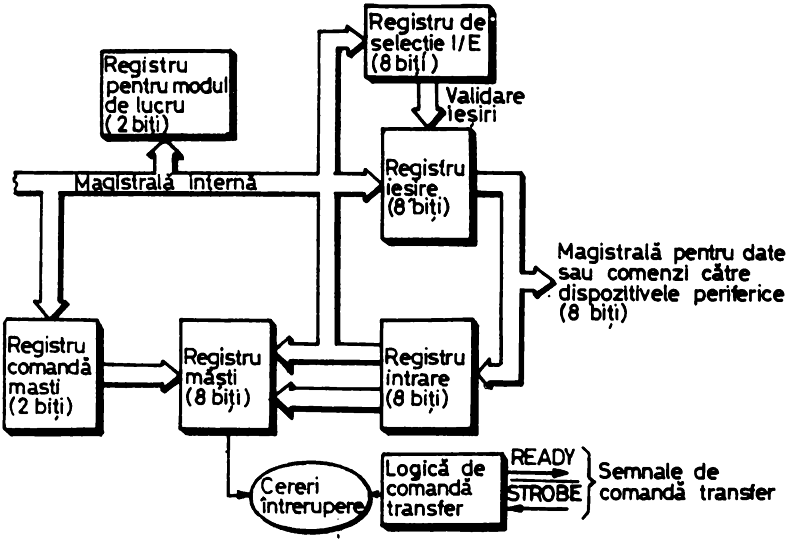


Fig. 2.19. Schema-bloc a circuitului PIO-Z80

Cele două *port*-uri de I/E, identice, au organizarea din figura 2.20, conţinând fiecare registre separate de intrare şi ieşire, patru registre de comandă şi partea de logică pentru controlul transferului. Toate transferurile între UC şi periferice utilizează registrele de intrare sau ieşire, schimbul efectiv de informaţie cu perifericul desfăşurându-se sub comanda semnalelor READY şi STROBE. Modul de lucru al *port*-ului este memorat într-un registru de doi biţi. Celelalte trei registre de comandă sunt utilizate numai în modul 3.

Registrul de selecţie I/E specifică biţii de ieşire, validându-i; biţii rămaşi sunt consideraţi intrări. Registrele de măşti şi de comandă-măşti se folosesc pentru fixarea condiţiilor de întrerupere. Registrul de măşti memorează biţii activi şi pe cei inactivi sau mascaţi. Cu ajutorul registrului de comandă-măşti se precizează întâi dacă starea activă a biţilor este "1" sau "0", apoi dacă întreruperea se va genera când oricare din biţii nemascaţi este activ, condiţia SAU, ori numai când toţi biţii de intrare nemascaţi sunt activi, condiţia ŞI.

Fig. 2.20. Organizarea unui *port* de I/E din PIO-Z80



Logica de comandă a întreruperilor asigură desfăşurarea protocolului de întrerupere al UC-Z80 în cazul conectării circuitului PIO-Z80 într-o structură în lanţ, *daisy-chain*. Într-o astfel de conectare prioritatea unui circuit e determinată de poziţia lui fizică. Transmiterea priorităţii se face cu ajutorul a două semnale, semnalul de intrare pentru validarea întreruperilor, IEI, şi semnalul de ieşire pentru validarea întreruperilor, IEO. Dispozitivul cel mai apropiat de UC este prioritar, iar în cadrul unui PIO-Z80 *port*-ul A este prioritar faţă de *port*-ul B. Întreruperile generate cu PIO-Z80 pot semnala unităţii centrale fie o cerere de transfer din partea perifericului, când se lucrează în modurile 0, 1 sau 2, fie îndeplinirea unei condiţii, o valoare programată a liniilor de I/E, când se lucrează în modul 3.

Comportarea în întreruperi a lui PIO-Z80 corespunde modului 2 de lucru al microprocesorului Z80; după acceptarea unei întreruperi PIO trebuie să genereze un vector de 8 biţi, care împreună cu conţinutul registrului I formează adresa unei locaţii de unde se va citi adresa subrutinei de serviciu. Fiecare *port* poate fi programat să aibă un vector de întrerupere propriu. Cel mai puţin semnificativ bit al vectorului este întotdeauna pus pe zero în PIO-Z80, având în vedere că adresa pe 16 biţi a subrutinei de serviciu trebuie să se găsească în două locaţii succesive de memorie începând cu o adresă pară.

Spre deosebire de alte circuite periferice din familia Z80, PIO nu validează întreruperile imediat după programare ci mai târziu, după ce a fost sesizată o trecere a lui $\overline{M1}$ pe "0", corespunzătoare, de exemplu, unui ciclu de extragere. Pentru a permite tratarea întreruperilor de la dispozitivele mai puţin prioritare PIO trebuie să genereze semnalul IEO cât mai repede după terminarea rutinei de serviciu. În acest scop PIO-Z80 ştie să decodifice direct de pe magistrala de date a sistemului instrucţiunea de revenire RETI şi în funcţie de starea sa, să poziţioneze semnalul IEO pe "1".

2.2.2. DESCRIEREA CONEXIUNILOR EXTERNE

PIO-Z80 are 40 de conexiuni externe grupate, aşa cum se vede în figura 2.21, pe funcţiuni, pe de o parte legate de interfaţa cu UC, magistrala de date, semnalele de comandă a circuitului şi a întreruperilor, iar pe de altă parte legate de interfaţa cu perifericele, *port*-urile A şi B. Circuitul se poate conecta direct cu UC-Z80 fără a fi necesară adăugarea de logică externă. În sisteme mari ar putea fi necesare decodificatoare de adresă şi/sau *buffer*-e.

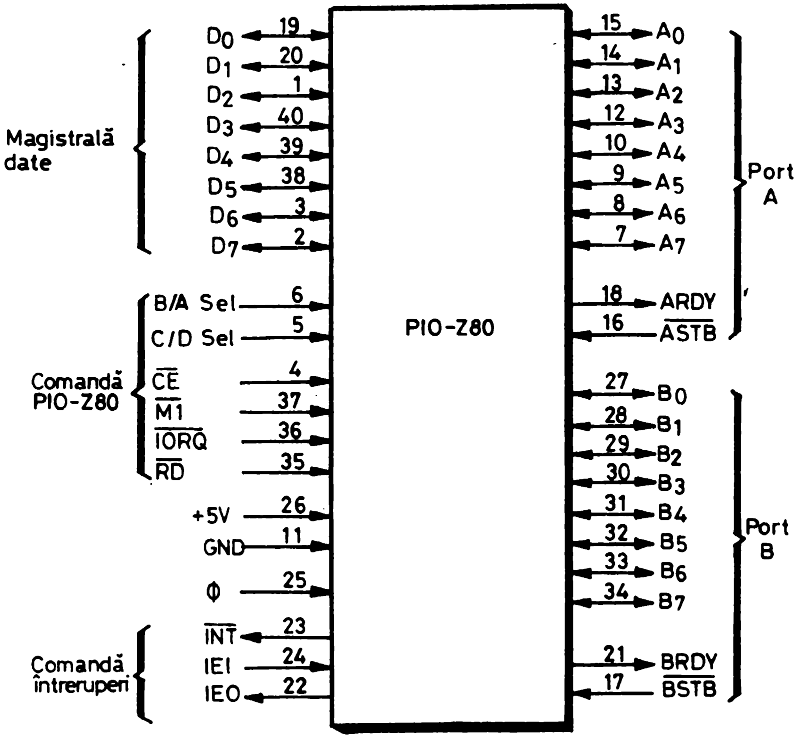


Fig. 2.21. Conexiunile externe ale circuitului PIO-Z80

2.2.2.1. Magistrala de date

$D_0 \div D_7$, *Data Bus*. Intrări/ieşiri trei-stări active pe "1". Magistrala de date e utilizată pentru a transfera date şi comenzi între UC şi PIO-Z80.

2.2.2.2. Semnalele de comandă a circuitului

$\overline{M1}$, *Machine Cycle One*, primul ciclu-maşină. Intrare de la UC activă pe "0". Este folosită ca impuls de sincronizare pentru a comanda diferite operaţii interne din PIO. $\overline{M1}$ activ împreună cu $\overline{RD}$ semnalează un ciclu de extragere,

iar împreună cu $\overline{IORQ}$ achitarea unei întreruperi. În afara acestor două funcţiuni cunoscute, $\overline{M1}$ mai serveşte la sincronizarea logicii de întreruperi şi, atunci când apare fără $\overline{RD}$ sau $\overline{IORQ}$ pe "0", la iniţializarea circuitului.

$\overline{RD}$, *Read Cycle Status*, ciclu de citire. Intrare de la UC, activă pe "0". Când este activă semnalează faptul că sistemul execută un ciclu de citire din memoria externă sau un ciclu de citire I/E. Împreună cu B/A Sel, C/D Sel, $\overline{CE}$ şi $\overline{IORQ}$ este întrebuinţată pentru a transfera date de la PIO-Z80 la UC.

$\overline{IORQ}$, *Input Output Request*, cerere de I/E. Intrare de la UC, activă pe "0". Activă odată cu $\overline{CE}$ şi $\overline{RD}$ semnalează o operaţie de citire: *port*-ul adresat de B/A Sel transferă datele către UC. Activă doar împreună cu $\overline{CE}$, semnalează o operaţie de scriere: în *port*-ul adresat de B/A Sel sunt scrise date sau comenzi în funcţie de starea liniei C/D Sel. $\overline{IORQ}$ şi $\overline{M1}$ active simultan specifică un ciclu de achitare întrerupere în care PIO va trebui să plaseze un vector pe magistrala de date.

$\overline{CE}$, *Chip Enable*, validare capsulă. Intrare activă pe "0". Activarea acestei intrări selectează circuitul pentru a se putea efectua transferuri de date cu UC.

B/A Sel, *Port B or A Select*, selecţie *port* A sau B. Intrare activă pe "1". Selectează unul din cele două *port*-uri: "0" pentru A, "1" pentru B. De obicei această intrare este comandată cu bitul A_0 al magistralei de adrese.

C/D Sel, *Control or Data Select*, selecţie comenzi sau date. Intrare activă pe "1". Defineşte tipul transferului ce se efectuează între UC şi PIO-Z80; "0" pentru date; "1" pentru comenzi. Intrarea e de obicei comandată cu bitul A_1 al magistralei de adrese.

2.2.2.3. Semnalele de comandă a întreruperilor

$\overline{INT}$, *Interrupt Request*, cerere întrerupere. Ieşire drenă-în-gol activă pe "0". Activarea acestei ieşiri înseamnă că PIO-Z80 emite o cerere de întrerupere către UC.

IEI, *Interrupt Enable In*, intrare validare întreruperi. Intrare activă pe "1". Semnal utilizat pentru conectarea circuitului într-un lanţ de priorităţi: IEI=1 indică circuitului PIO-Z80 faptul că nici un dispozitiv prioritar nu este servit în acel moment de o rutină de tratare a întreruperii.

IEO, *Interrupt Enable Out*, ieşire validare întreruperi. Ieşire activă pe "1". Această ieşire se conectează la intrarea IEI a următorului circuit din lanţul de priorităţi. IEO este "1" numai dacă intrarea IEI este "1" şi UC nu serveşte o întrerupere generată de PIO. În acest fel IEO poate bloca generarea întreruperilor de către dispozitivele cu prioritate scăzută pe timpul tratării unei întreruperi solicitate de un circuit prioritar.

2.2.2.4. *Port*-ul A

$A_0 \div A_7$, *Port A Bus*, magistrală *port* A. Intrări/ieşiri trei-stări prin intermediul cărora se transferă date, stări sau comenzi între PIO-Z80 şi echipamentul periferic conectat.

ARDY, *Register A Ready*, registru A gata. Ieşire activă pe "1" a cărei semnificaţie depinde de modul de lucru programat:

Ieşire-octet. Este activată când registrul de ieşire al *port*-ului A a fost încărcat, magistrala cu perifericul e stabilă şi pregătită pentru transferul octetului la periferic.

Intrare-octet. ARDY e activată când registrul de intrare al *port*-ului A este gol şi gata să accepte date de la periferic.

Intrare/ieşire-octet. Ieşirea este activată când datele sunt disponibile în registrul de ieşire al *port*-ului A pentru a fi transferate către periferic. În acest mod datele nu vor fi plasate pe magistrala cu perifericul până când $\overline{ASTB}$ nu e activ.

Intrare/ieşire pe bit. ARDY e invalidat şi forţat pe "0".

$\overline{ASTB}$, *Port A Strobe Pulse From Peripheral Device*, impuls de eşantionare de la periferic pentru *port*-ul A. Intrare activă pe "0" având următoarele semnificaţii, în funcţie de modul de lucru:

Ieşire-octet. Frontul pozitiv al acestui semnal este generat de periferic pentru a semnala recepţionarea datelor emise de PIO.

Intrare-octet. Se emite de periferic pentru a încărca datele în registrul de intrare al *port*-ului A. Încărcarea se face când semnalul este activ.

Intrare/ieşire-octet. Când semnalul este activ datele din registrul de ieşire al *port*-ului A sunt emise pe magistrala cu perifericul *port*-ului A. Frontul pozitiv al semnalului înseamnă recepţionarea datelor.

Intrare/ieşire pe bit. Intrarea $\overline{ASTB}$ se inhibă intern.

2.2.2.5. *Port*-ul B

$B_0 \div B_7$, *Port B Bus*, magistrală *port* B. Intrări/ieşiri trei-stări destinate transferării de date, stări sau comenzi între PIO-Z80 şi echipamente periferice. Ieşirile acestui *port* pot comanda tranzistori Darlington asigurând un curent de 1,5 mA la 1,5 V.

BRDY, *Register B Ready*, registru B gata. Ieşire activă pe "1" cu funcţii similare cu cele ale ieşirii ARDY. Menţionăm că atunci când se lucrează în mod bidirecţional, numai cu *port*-ul A, semnalul BRDY este "1" dacă registrul de intrare al *port*-ului A este gol şi pregătit să accepte date de la periferic.

$\overline{BSTB}$, *Port B Strobe Pulse From Peripheral Device*, impuls de eşantionare de la periferic pentru *port*-ul B. Intrare activă pe "0" cu funcţii similare cu cele ale intrării $\overline{ASTB}$. Numai când se lucrează în modul bidirecţional semnalul

$\overline{\text{BSTB}}$ eşantionează datele de la periferic înscriindu-le în registrul de intrare al *port*-ului A.

2.2.3. DIAGRAME DE TIMP

2.2.3.1. **Ciclul de scriere UC**

Scrierea comenzilor sau datelor în PIO-Z80 se face cu ajutorul instrucţiunilor de ieşire. Diagrama de timp a unui ciclu de scriere este dată în figura 2.22. Se observă concordanţa cu ciclul de ieşire al microprocesorului Z80 (vezi figura 2.10) care introduce în mod automat o singură stare de aşteptare T_w. Pentru a se asigura funcţionarea corectă a circuitului PIO, nu se mai admite inserarea altor stări T_w. Menţionăm, de asemenea, că neexistând o conexiune externă pe care să primească un semnal explicit de comandă a scrierii, PIO-Z80 îl generează intern. Acest semnal, $\overline{\text{WR}}^*$ în figura 2.22, generat intern, este activ când lipseşte $\overline{\text{RD}}$.

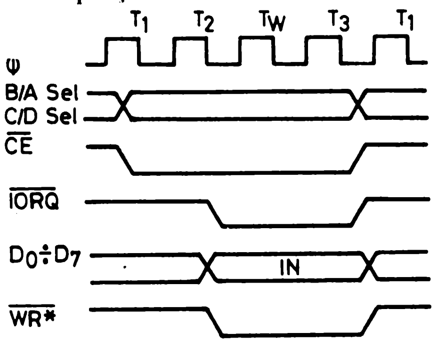


Fig. 2.22.
Ciclul de scriere UC în PIO-Z80

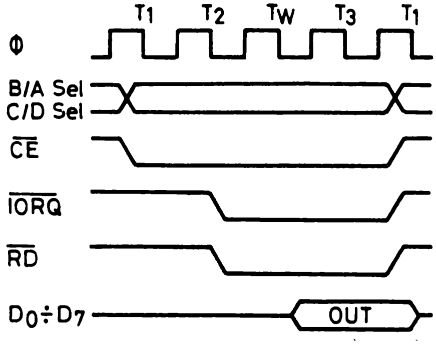


Fig. 2.23.
Ciclul de citire UC din PIO-Z80

2.2.3.2. **Ciclul de citire UC**

Diagrama de timp, prezentată în figura 2.23, arată, ca şi în cazul scrierii, concordanţa cu ciclul de intrare al microprocesorului Z80. La fel, nu se permite introducerea de stări T_w suplimentare faţă de starea adăugată în mod automat de UC-Z80.

2.2.3.3. **Modul 0 (ieşire-octet)**

Desfăşurarea în timp a operaţiilor specifice modului 0 de lucru, ieşire-octet, este ilustrată în figurile 2.24, 2.25 şi 2.26. Iniţierea unui ciclu de ieşire se face întotdeauna prin execuţia, de către UC, a unei instrucţiuni OUT.

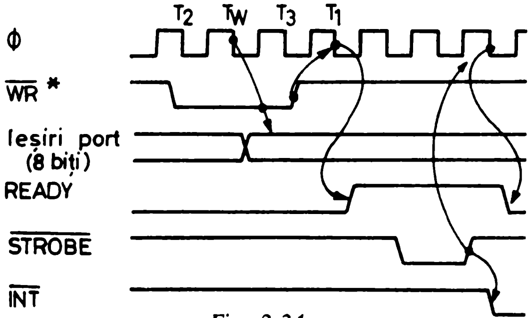


Fig. 2.24.
Ieşire-octet utilizând ambele semnale READY şi $\overline{\text{STROBE}}$

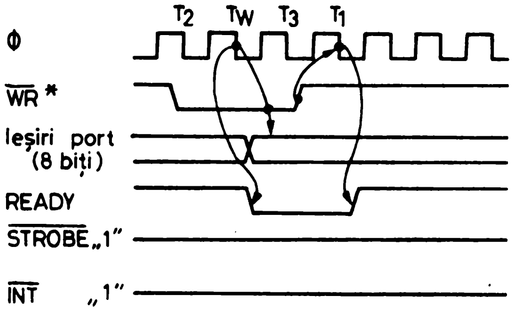


Fig. 2.25.
Ieşire-octet cu intrarea $\overline{\text{STROBE}}$ conectată la "1"

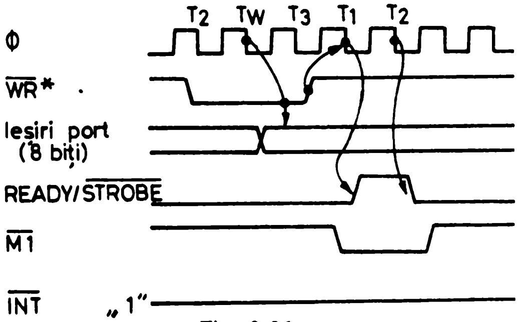


Fig. 2.26.
Ieşire-octet cu liniile READY şi $\overline{\text{STROBE}}$ legate împreună

Impulsul $\overline{\text{WR}}^*$, generat intern pe timpul unei astfel de instrucţiuni, va condiţiona înscrierea datelor trimise pe magistrala de date a sistemului, $D_0 \div D_7$, în registrul de ieşire al *port*-ului adresat. La sfârşitul instrucţiunii de ieşire, după dezactivarea lui $\overline{\text{WR}}^*$, PIO va poziţiona pe "1", cu frontul negativ al ceasului Φ, ieşirea READY corespunzătoare *port*-ului în care s-a făcut scrierea. În acest fel se indică perifericului că poate prelua un octet, frontul pozitiv al semnalului READY putând fi folosit de către dispozitivul de I/E pentru memorarea acestui octet. READY va rămâne activ, "1", până la recepţionarea unui front pozitiv pe linia $\overline{\text{STROBE}}$ cu ajutorul căruia se va genera totodată, o întrerupere, dacă *port*-ul respectiv a fost programat şi validat să lucreze în întreruperi si dacă este prioritar.

În cazul în care se lucrează fără semnalul $\overline{\text{STROBE}}$, cu intrarea corespunzătoare legată la "1", ieşirea READY va fi forţată pe "0" după o perioadă şi jumătate a lui Φ de la activarea comenzii $\overline{\text{IORQ}}$. READY va fi repus pe "1" cu primul front negativ al ceasului după dezactivarea lui $\overline{\text{IORQ}}$ (figura 2.25). Această acţionare garantează că READY este întotdeauna pe "0" la schimbarea datelor în *port* şi că trecerea lui pe "1" se face ori de câte ori se va executa o instrucţiune OUT.

Dacă cele două linii de comandă a transferului, READY şi $\overline{\text{STROBE}}$, sunt legate împreună, PIO-Z80 va genera pe aceste linii un impuls cu durata o perioadă a ceasului Φ ca în figura 2.26. Frontul pozitiv al lui READY/$\overline{\text{STROBE}}$, nu va mai poziţiona pe "0" ieşirea $\overline{\text{INT}}$, datorită logicii interne din PIO, care nu permite ca durata impulsului pozitiv pe $\overline{\text{STROBE}}$ să fie mai mică decât $\overline{\text{M1}}$.

2.2.3.4. Modul 1 (intrare-octet)

Un ciclu de intrare poate fi inițiat de către periferic prin punerea pe "0" a liniei $\overline{STROBE}$ corespunzătoare *port*-ului programat să lucreze în modul 1, ca în figura 2.27. Pe timpul cât acest semnal este "0" intrările în *port* sunt eșantionate, transferându-se în registrul de intrare al *port*-ului. Cu frontul pozitiv al lui $\overline{STROBE}$ se poate genera o întrerupere, ca și în modul 0, dacă *port*-ul respectiv a fost programat și validat să lucreze în întreruperi și dacă este prioritar. După trecerea pe "1" a lui $\overline{STROBE}$, cu următorul front negativ al lui Φ, se va inactiva ieșirea READY semnalând perifericului că registrul de intrare este plin și că trebuie inhibată încărcarea unui nou octet, până la preluarea de către UC a octetului păstrat în registrul de intrare al *port*-ului. Preluarea se face cu ajutorul unei instrucțiuni IN, de obicei în timpul rutinei de tratare a întreruperii. Activarea semnalului READY se va face la sfârșitul acestei instrucțiuni, cu primul front negativ, al ceasului Φ după trecerea pe "1" a lui $\overline{RD}$, indicându-se astfel dispozitivului de I/E că poate transmite un nou octet către PIO.

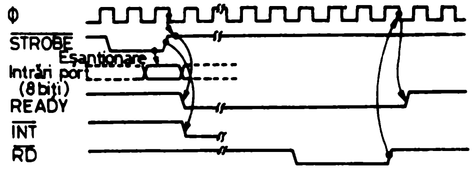


Fig. 2.27. Intrare-octet utilizând ambele semnale READY și $\overline{STROBE}$

Ținând cont că după inițializare ieșirea READY rămâne pe "0", pentru a lansa un transfer în modul 1 este necesară, în unele sisteme, execuția unei instrucțiuni IN false pentru a se face READY=1 [27].

Dacă nu se folosește, intrarea $\overline{STROBE}$ trebuie conectată la "0". În această situație semnalul READY va fi inactivat o perioadă și jumătate a ceasului Φ după trecerea pe "0" a comenzii $\overline{IORQ}$ în timpul unei instrucțiuni IN de citire a *port*-ului (figura 2.28). Astfel se previne schimbarea datelor în registrul de intrare pe timpul eșantionării acestuia de către UC. READY va fi reactivat la fel ca mai sus cu primul front negativ al lui Φ după trecerea pe "1" a lui $\overline{RD}$.

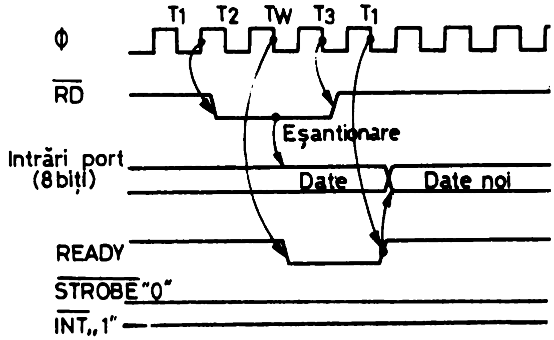


Fig. 2.28. Intrare-octet cu linia $\overline{STROBE}$ conectată la "0"

2.2.3.5. Modul 2 (intrare/ieșire-octet)

În figura 2.29 este dată desfășurarea în timp a transferului de octeți când PIO-Z80 este programat să lucreze în modul 2. Acest mod poate fi programat

numai pentru *port*-ul A, utilizându-se toate semnalele de comandă-transfer ale circuitului PIO.

După cum se poate vedea în figura 2.29, funcţionarea este aproape identică cu funcţionarea corespunzătoare modurilor 0 şi 1, descrisă anterior (vezi figurile 2.24, 2.27): pentru comanda ieşirii se folosesc liniile ARDY şi $\overline{ASTB}$, iar pentru intrare BRDY şi $\overline{BSTB}$. Diferenţa constă numai în faptul că, atunci când se lucrează în modul 2, ieşire-octet, datele sunt plasate pe magistrala către periferic numai pe durata cât $\overline{ASTB}$ este activ, dispozitivul de I/E putând întrebuinţa frontul pozitiv al acestui semnal pentru memorarea octetului.

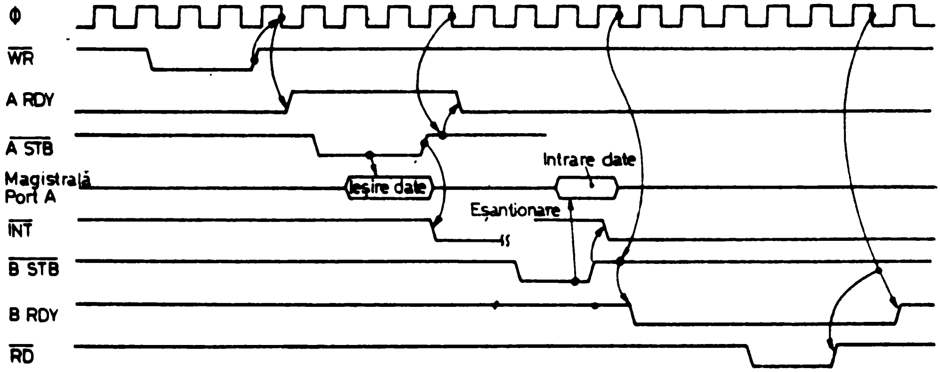


Fig. 2.29. Intrare/ieşire-octet

În timp ce $\overline{ASTB}$ este activ, adică atunci când PIO efectuează o ieşire-octet, dispozitivul periferic nu trebuie să plaseze date pe magistrala de intrare/ieşire a *port*-ului A. În scopul rezolvării acestui conflict potenţial perifericul poate utiliza semnalul $\overline{BSTB}$ decalat în timp faţă de $\overline{ASTB}$, pentru a valida emisia datelor pe magistrală. Această condiţionare este permisă ţinând seama că PIO-Z80, atunci când lucrează în modul 2, memorează datele în registrul de intrare când $\overline{BSTB}=0$, pe nivel. De asemenea, validarea pe magistrală a datelor cu $\overline{BSTB}$ este permisă şi pentru că PIO-Z80 nu impune nici un timp de menţinere a datelor după dezactivarea lui $\overline{BSTB}$. Dacă $\overline{ASTB}$ este activ pe timpul cât UC execută o operaţie de citire a *port*-ului A, ca răspuns la o întrerupere generată cu $\overline{BSTB}$, UC va citi registrul de ieşire în locul registrului de intrare. Deşi datele eşantionate cu $\overline{BSTB}$ sunt, în această situaţie, corect memorate în registrul de intrare al *port*-ului, UC nu poate citi registrul. Pentru a elimina această situaţie, activarea lui $\overline{ASTB}$ poate fi condiţionată şi de BRDY=1, având în vedere că, atunci când BRDY=0, PIO-Z80 poate aştepta rezolvarea de către UC a unei întreruperi generate cu $\overline{BSTB}$.

2.2.3.6. **Modul 3 (intrare/ieşire pe bit)**

Când PIO-Z80 funcţionează în modul 3 semnalele de comandă a transferului nu sunt folosite, unitatea centrală a sistemului putând să execute în orice

moment operaţii de scriere sau citire în/din *port*-uri. La scriere datele sunt memorate în registrul de ieşire cu aceeaşi desfăşurare în timp ca în modul 0. Aşa cum am spus la descrierea conexiunilor externe, ARDY este întotdeauna forţat pe "0" când *port*-ul A lucrează în modul 3. Acelaşi lucru se întâmplă şi cu BRDY, excepţie făcând situaţia când *port*-ul A este în modul 2, ieşirea BRDY rămânând în acest caz neafectată.

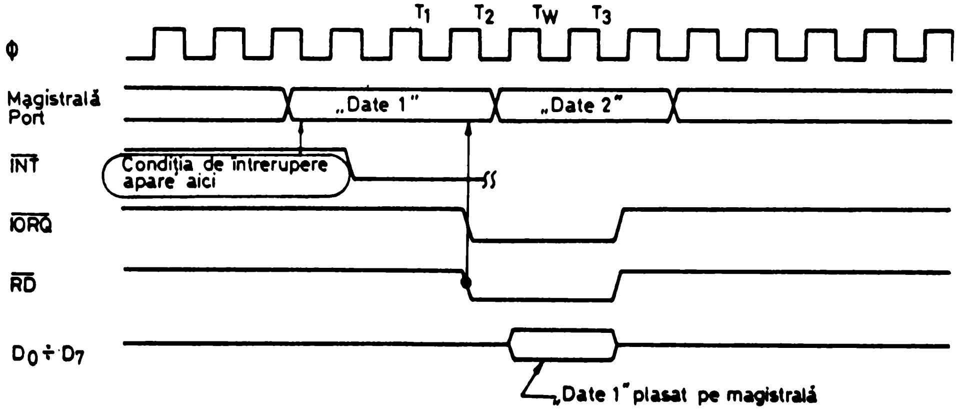


Fig. 2.30. Citire dintr-un *port* programat în modul 3

În figura 2.30 este dată desfăşurarea în timp a unei operaţii de citire a unui *port* programat să lucreze în modul 3. Octetul citit de către UC va fi compus din biţi aparţinând registrului de ieşire, pentru liniile desemnate ca ieşiri, şi din biţi aparţinând registrului de intrare, pentru liniile specificate ca intrări. Registrul de intrare va conţine datele prezente pe magistrala cu perifericul înainte de frontul negativ al semnalului $\overline{RD}$; în figura 2.30 UC va citi "Date 1", condiţia care a generat întreruperea, şi nu "Date 2".

Când se lucrează în modul 3 PIO supraveghează în permanenţă liniile nemaseate ale *port*-ului pentru a surprinde apariţia unei configuraţii de biţi stabilite să genereze o întrerupere, evident numai dacă întreruperile corespunzătoare *port*-ului respectiv sunt validate. Întreruperea se generează atunci când condiţia logică se schimbă din falsă în adevărată. Pentru ca acelaşi *port* să genereze o nouă întrerupere, condiţia care a generat întreruperea trebuie să devină falsă şi apoi, din nou, adevărată. De exemplu, să presupunem condiţia logică SAU. Întreruperea va fi generată dacă o linie nemascată devine activă. Dacă, în continuare, o altă linie nemascată devine şi ea activă, PIO-Z80 ***nu va mai genera o nouă întrerupere***. Mai mult, dacă două linii nemascate devin active simultan, PIO va genera o singură întrerupere. Înainte ca PIO să genereze o altă întrerupere trebuie, deci, ca toate liniile nemascate să devină inactive şi apoi cel puţin una să fie activată. Conexiunile externe ale *port*-ului definite ca ieşiri pot contribui la condiţia logică ce generează întreruperea dacă poziţiile lor nu sunt mascate.

În cazul în care condiția logică devine adevărată puțin înainte de $\overline{M1}$ sau pe durata lui $\overline{M1}$, întreruperea va fi generată după frontul pozitiv al acestui semnal, dacă această condiție rămâne adevărată și după front.

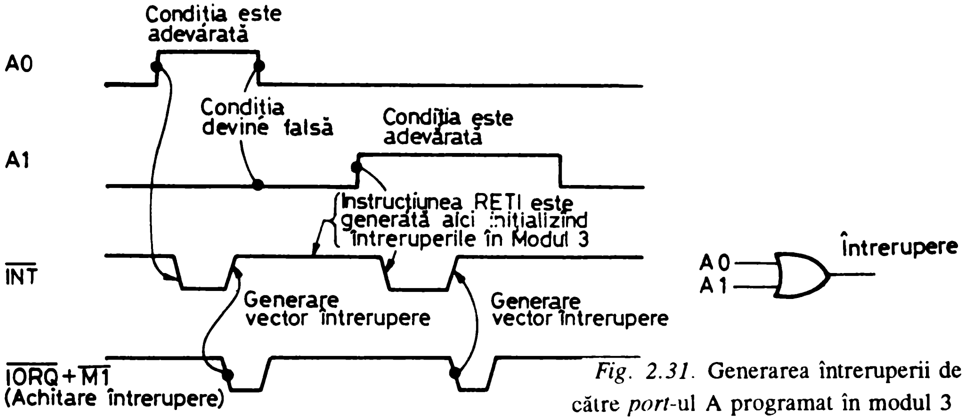


Fig. 2.31. Generarea întreruperii de către *port*-ul A programat în modul 3

În figura 2.31 se dă un exemplu de generare a întreruperii de către *port*-ul A programat să lucreze în modul 3. Condiția logică selectată este SAU iar starea activă "1". Toți biții sunt mascați în afară de A_0 și A_1, ceea ce echivalează deci cu o poartă SAU în logică pozitivă cu două intrări. După poziționarea lui A_0 pe "1", condiția logică devine adevărată, PIO generând o întrerupere. UC răspunde la această întrerupere cu un ciclu de achitare-întrerupere, $\overline{IORQ}+\overline{M1}=0$, în care PIO va transmite vectorul de întrerupere și UC va apela subrutina de serviciu corespunzătoare. A_0 poate trece acum pe "0" fie singur, fie ca rezultat a unei acțiuni întreprinse în subrutina de serviciu.

Trecerea lui A_0 pe "0" înseamnă, în situația din figura 2.31 când și A_1 e "0", o anulare a condiției logice care generează întreruperea. Acum, după execuția instrucțiunii de revenire RETI care inițializează schema de întreruperi din PIO, o nouă întrerupere poate fi activată, de exemplu, prin poziționarea lui A_1 pe "1".

Din acest exemplu reținem deci, în primul rând, că pentru ca A_1 să genereze o întrerupere, semnalul nu trebuie să devină "1" decât după ce A_0 a trecut pe "0". De asemenea, pentru ca A_1 să genereze o întrerupere, va trebui să se poziționeze pe "1" după execuția instrucțiunii RETI din subrutina de tratare a întreruperii generate de A_0. În concluzie, pentru a se genera o nouă întrerupere, condiția logică trebuie să devină falsă după achitarea întreruperii precedente și să redevină adevărată după ce RETI inițializează schema de întreruperi din PIO-Z80.

2.2.3.7. **Întreruperi**

Pentru o mai bună înțelegere a comportării în întreruperi a circuitului PIO-Z8O dăm în figura 2.32 schema generală a blocului de comandă a întreru-

perilor implementat, cu mici adaptări, în toate circuitele periferice ale familiei Z80. Această schemă poate fi utilizată şi în cazul interfaţării microprocesorului Z80 cu circuite periferice care nu au prevăzut mecanismul de lucru în întreruperi specific familiei Z80 (modul 2).

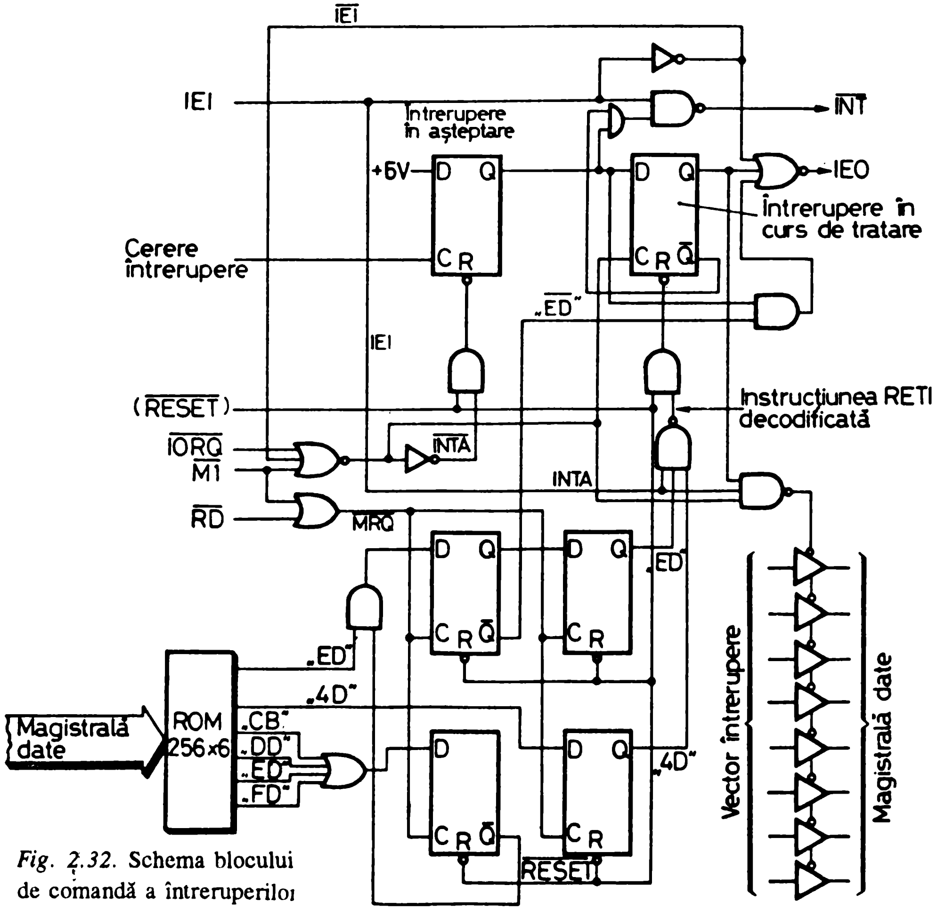


Fig. 2.32. Schema blocului de comandă a întreruperilor

Funcţiile realizate de blocul de comandă a întreruperilor sunt: activarea semnalului de întrerupere $\overline{INT}$ către unitatea centrală, comanda ieşirii IEO, plasarea pe magistrala de date a vectorului în ciclul de achitare a întreruperii, decodificarea instrucţiunii RETI în vederea reiniţializării schemei. Corespunzător acestor funcţii, blocul este alcătuit din bistabilii "Întrerupere în aşteptare" şi "Întrerupere în curs de tratare", alţi patru bistabili şi o memorie ROM pentru decodificarea instrucţiunii RETI, circuite combinaţionale de comandă.

La iniţializarea circuitului bistabilii se şterg, ceea ce conduce la dezactivarea liniei $\overline{INT}$ şi la IEO=IEI. Pentru generarea unei întreruperi spre UC este necesar, mai întâi, să se activeze semnalul intern "Cerere-întrerupere" specific

fiecărui circuit periferic din familia Z80. Activarea acestui semnal conduce, după cum se vede în figura 2.32, la poziţionarea pe "1" a bistabilului "Întrerupere în aşteptare". Se observă, de asemenea, că poziţionarea pe "1" a bistabilului se inhibă la dispozitive prioritare, cu IEI = 1, pe durata unui ciclu de achitare. În acest fel se asigură îngheţarea lanţului de priorităţi spre prioritatea înaltă pe timpul unui ciclu de achitare a întreruperii. În continuare, pentru ca dispozitivul să activeze semnalul $\overline{INT}$, trebuie îndeplinite simultan condiţiile: IEI = 1, bistabilul "Întrerupere în aşteptare" să fie pe "1", bistabilul "Întrerupere în curs de tratare" pe "0". Totodată poziţionarea pe "1" a bistabilului "Întrerupere în aşteptare" duce la punerea pe zero a ieşirii IEO. UC răspunde la întrerupere cu un ciclu de achitare în care $\overline{IORQ} = \overline{M1} = 0$. Într-un lanţ de priorităţi pot exista mai multe circuite cu bistabili "Întrerupere în aşteptare" poziţionaţi pe "1", unul singur având însă IEI = 1 şi IEO = 0. La acest dispozitiv un ciclu de achitare face ca bistabilul "Întrerupere în aşteptare" să fie şters şi bistabilul "Întrerupere în curs de tratare" să fie pus pe "1". Pe durata ciclului de achitare, dispozitivul cu întreruperea în aşteptare prioritar îşi plasează vectorul de întrerupere pe magistrala de date. La ştergerea bistabilului "Întrerupere în aşteptare" şi trecerea pe "1" a "Întreruperii în curs de tratare" linia $\overline{INT}$ se dezactivează. Pentru reiniţializarea schemei de întreruperi blocul de comandă supraveghează în permanenţă magistrala de date cu scopul de a surprinde extragerea celor doi octeţi care formează codul-operaţie al instrucţiunii RETI, 0EDH şi 4DH. În momentul decelării acestei situaţii, bistabilul "Întrerupere în curs de tratare" este şters, permiţându-se astfel generarea unei noi întreruperi fie de către dispozitivul însuşi, fie de un dispozitiv cu prioritate scăzută. Mai observăm că IEI intervine şi în ştergerea bistabilului "Întrerupere în curs de tratare" asigurând ca acesta să fie pus pe "0" de instrucţiunea RETI numai la dispozitivul prioritar. Aceasta impune însă ca IEO să treacă temporar pe "1" la dispozitivele prioritare cu întreruperi în aşteptare, când se decodifică primul octet al codului-operaţie al instrucţiunii RETI, pentru a permite ştergerea bistabilului "Întrerupere în curs de tratare" într-un dispozitiv cu prioritate scăzută.

În concluzie, pentru ca un dispozitiv să activeze linia $\overline{INT}$, este necesar să fie îndeplinite următoarele condiţii:

- IEI să fie "1";
- bistabilul "Întrerupere în aşteptare" să fie pe "1";
- bistabilul "Întrerupere în curs de tratare" să fie pe "0".

Pentru ca ieşirea IEO să fie pe "1" trebuie îndeplinite condiţiile:

- IEI să fie "1";
- bistabilul "Întrerupere în curs de tratare" să fie "0";
- bistabilul "Întrerupere în aşteptare" să fie "0" sau primul octet, 0EDH, al codului-operaţie al instrucţiunii RETI să fie prezent pe magistrala de date.

Achitarea unei întreruperi generate de PIO-Z80 se face, conform procedurii specifice microprocesorului Z80, prin activarea de către unitatea

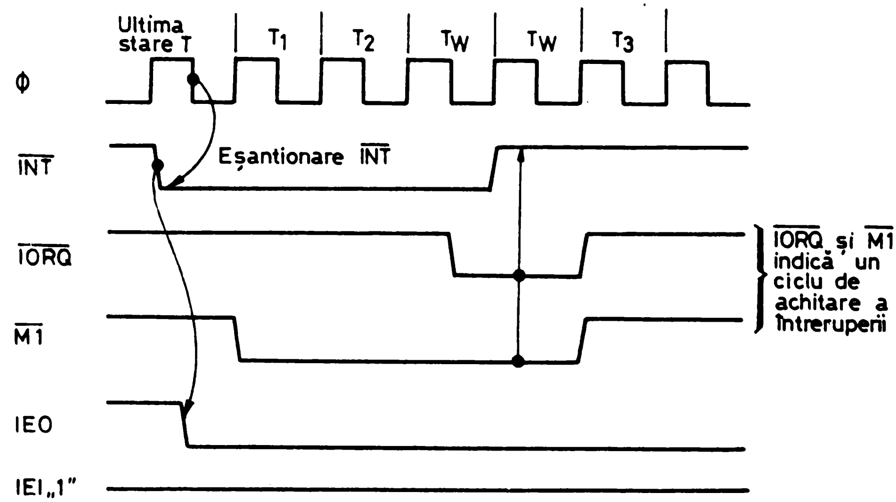


Fig. 2.33. Achitarea unei întreruperi generate de PIO-Z80

centrală a semnalelor $\overline{M1}$ şi $\overline{IORQ}$ (figura 2.33 şi figura 2.13). Pe timpul cât aceste semnale sunt active, timp mărit prin introducerea celor două stări T_w, logica de întreruperi din PIO determină *port*-ul prioritar care a generat întreruperea şi plasează pe magistrala de date conţinutul registrului vector-întrerupere. Pentru a asigura stabilitatea lanţului IEI-IEO pe durata cât $\overline{IORQ}$ şi $\overline{M1}$ sunt "0", dispozitivele de I/E, aici *port*-urile din PIO-Z80, nu activează linia de întrerupere. În acest timp semnalele IEI-IEO se pot propaga prin maximum patru circuite PIO.

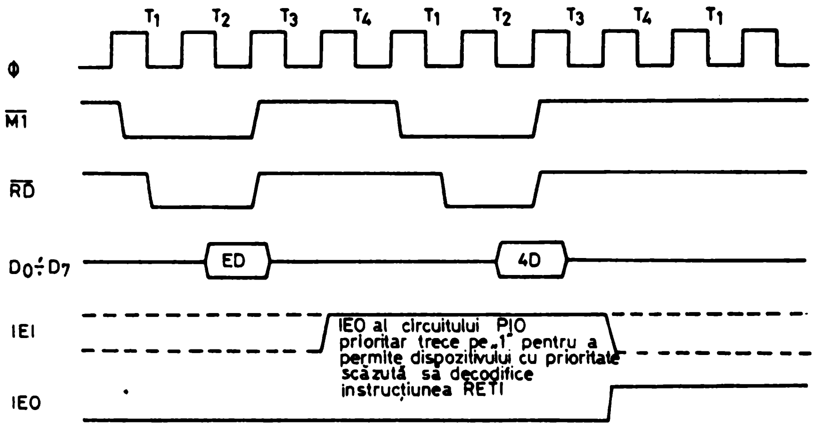


Fig. 2.34. Semnalele IEI-IEO în timpul decodificării instrucţiunii RETI

Un *port* care nu are nici o întrerupere în aşteptare va avea IEI=IEO. *Port*-ul prioritar, care a generat o întrerupere, va avea, în momentul achitării,

IEI=1 şi IEO=0. După ce această întrerupere a fost achitată, întreruperea urmând să fie deci tratată în subrutina de serviciu, IEO va rămâne "0" până la execuţia instrucţiunii RETI, inhibând activarea unei alte întreruperi de către dispozitivele cu prioritate scăzută. *Port*-urile prioritare, care au întreruperi în aşteptare neachitate încă, au IEO=0 până la decodificarea primului octet, OEDH, al instrucţiunii RETI (figura 2.34). În această situaţie, semnalul IEO al circuitului prioritar va trece temporar pe "1", permiţând dispozitivului cu prioritate scăzută având întreruperea în curs de tratare, ca, decodificând al doilea octet al codului-operaţie al instrucţiunii RETI, să-şi şteargă bistabilul "Întrerupere în curs de tratare" şi să treacă propriul semnal IEO pe "1".

2.2.4. PROGRAMAREA CIRCUITULUI PIO-Z80

Prin programarea lui PIO-Z80 se specifică următoarele trei grupe importante de parametri ce caracterizează funcţionarea ulterioară a circuitului:

- vectorul de întrerupere;
- modul de lucru;
- cuvântul de comandă-întreruperi.

2.2.4.1. **Vectorul de întrerupere**

PIO-Z80 a fost proiectat să lucreze conform protocolului corespunzător modului 2 de lucru în întreruperi al microprocesorului Z80 (vezi § 2.1). Acest protocol impune ca dispozitivul care întrerupe să genereze un vector de întrerupere, utilizat apoi de unitatea centrală ca octet mai puţin semnificativ al adresei de unde se va citi adresa subrutinei de serviciu. Vectorul de întrerupere este plasat pe magistrala de date, în timpul unui ciclu de achitare întrerupere, aşa cum am arătat în § 2.2.3. Vectorul de întrerupere dorit se încarcă în PIO prin scrierea în *port*-ul ales a unui cuvânt de comandă ce are următorul format:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	0

$D_0=0$ într-un cuvânt de comandă, semnifică circuitului PIO-Z80 că octetul respectiv este un vector de întrerupere şi deci că biţii $V_7 \div V_1$ trebuie încărcaţi în registrul intern corespunzător. La un ciclu de achitare-întrerupere, vectorul va apărea pe magistrala de date la fel ca mai sus, cu bitul cel mai puţin semnificativ pe "0". Reamintim că adresele de început ale subrutinelor de tratare a întreruperilor vor trebui plasate întotdeauna la adrese de cuvânt, pare.

2.2.4.2. **Modul de lucru**

Modul de lucru se stabileşte prin scrierea la *port*-ul ales din PIO-Z80 a unui cuvânt de comandă având formatul următor:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
M_1	M_0	X	X	1	1	1	1

Identificarea cuvântului de comandă utilizat pentru transmiterea modului de lucru se face cu ajutorul biţilor $D_3 \div D_0$, care trebuie să fie 1 1 1 1. Cei mai semnificativi biţi, M_1 şi M_0 stabilesc modul, după cum urmează:

M_1	M_0	Modul
0	0	0 (ieşire-octet)
0	1	1 (intrare-octet)
1	0	2 (intrare/ieşire-octet, bidirecţional)
1	1	3 (intrare/ieşire pe bit)

Biţii D_5 şi D_4 sunt ignoraţi, putând avea, deci, orice valoare.

Dacă se selectează modul 3, următorul cuvânt de comandă transmis către PIO trebuie să definească liniile *port*-ului ca intrări sau ieşiri. Formatul acestui cuvânt de comandă este:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
I/E_7	I/E_6	I/E_5	I/E_4	I/E_3	I/E_2	I/E_1	I/E_0

$I/E_i=1$ defineşte linia i a *port*-ului ca intrare iar $I/E_i=0$ ca ieşire.

2.2.4.3. **Cuvântul de comandă-întreruperi**

Comanda întreruperilor se face pentru fiecare din *port*-uri cu un cuvânt specific al cărui format este:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Validare întreruperi	ŞI/SAU	"1"/"0"	Urmează măştile	0	1	1	1

Cuvântul de comandă întreruperi se recunoaşte după cei mai puţin semnificativi patru biţi care trebuie să fie 0 1 1 1. Dacă bitul $D_7=1$, bistabilul intern de validare a întreruperilor pentru *port*-ul respectiv este pus pe "1" şi *port*-ul poate genera întreruperi. Dacă $D_7=0$, acest bistabil este şters şi *port*-ul nu mai poate activa linia de întreruperi. Dacă o întrerupere apare în timp ce se transmite cuvântul de comandă-întreruperi cu $D_7=0$ ea va fi memorată intern în PIO şi transmisă către UC după revalidare cu $D_7=1$. Biţii D_6, D_5 şi D_4 sunt utilizaţi în general numai în cazul selectării modului 3 de lucru. Poziţionarea pe "1" a lui D_4 într-un cuvînt de comandă-întreruperi, transmis în timp ce *port*-ul ales lucrează în oricare din cele patru moduri, va conduce la ştergerea unei eventuale întreruperi în aşteptare. Pentru precizarea condiţiilor de funcţionare ale *port*-ului selectat în modul 3, biţii $D_6 \div D_4$ trebuie programaţi astfel: D_6 defineşte funcţia logică ce conduce la generarea întreruperii, $D_6=1$ specifică funcţia ŞI iar $D_6=0$ funcţia SAU; D_5 precizează polaritatea activă a liniilor *port*-ului: $D_5=1$ înseamnă că liniile sunt active pe "1" iar $D_5=0$ pe "0"; $D_4=1$ înseamnă că următorul cuvânt transmis către PIO va fi un cuvânt de definire a măştilor, cuvânt ce are formatul următor:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
MB_7	MB_6	MB_5	MB_4	MB_3	MB_2	MB_1	MB_0

Numai liniile *port*-ului ai căror biţi de mascare, MB_i, sunt zero, vor fi luate în consideraţie la generarea unei întreruperi.

Bistabilul de validare a întreruperilor unui *port* poate fi pus pe "1" sau pe "0" separat, fără a modifica acţiunea cuvântului de comandă-întreruperi precedent, prin transmiterea unei comenzi cu formatul:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Validare întreruperi	X	X	X	0	0	1	1

Această comandă este, evident, asincronă cu generarea întreruperii de către *port*, generare care poate să apară chiar pe timpul cât UC transmite octetul de invalidare la PIO; *port*-ul generează o întrerupere achitată de unitatea centrală, dar vectorul nu se mai transmite pe durata ciclului de achitare datorită comenzii de invalidare. Ca rezultat va apărea o eroare de program. Pentru a rezolva această situaţie programatorul trebuie să invalideze temporar întreruperile pe durata transmiterii comenzii spre PIO-Z80, ca în secvenţa de mai jos:

```
...
LD A,03H        ; Încărcare A cu comanda de invalidare
DI              ; Invalidare întreruperi UC
OUT (PIO),A     ; Transmitere comandă către PIO
EI              ; Validare întreruperi UC
...
```

2.2.4.4. Inițializarea circuitului

La punerea sub tensiune PIO-Z80 se inițializează după cum urmează:

– registrele pentru măști ale ambelor *port*-uri sunt puse pe "1", inhibându-se astfel toți biții;

– este selectat modul 1 de lucru, intrare-octet, liniile: $A_0 \div A_7$, $B_0 \div B_7$ fiind trecute în starea de impedanță mare, iar ieșirile ARDY și BRDY pe "0";

– registrele de ieșire ale celor două *port*-uri sunt puse pe "0";

– bistabilii de validare a întreruperilor din ambele *port*-uri sunt puși pe "0", invalidându-se astfel generarea întreruperilor; de subliniat că registrele care păstrează vectorii de întrerupere nu se inițializează.

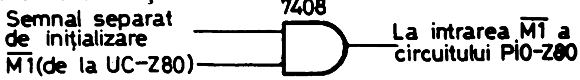


Fig. 2.35. Inițializarea hardware a lui PIO-Z80

În afară de această inițializare la punerea sub tensiune, PIO-Z80 mai poate fi inițializat prin activarea intrării $\overline{M1}$ cu condiția ca $\overline{RD}$ și $\overline{IORQ}$ să rămână pe "1". După ce $\overline{M1}$ va trece pe "1" circuitul intră în starea inițială descrisă mai sus. Acest mod de inițializare hardware, fără linie separată de ștergere, a fost impus de limitările datorate necesității de a împacheta *chip*-ul într-o capsulă cu 40 de conexiuni externe. Astfel, se permite ștergerea circuitului cu ajutorul unui semnal separat de inițializare, activ pe "0", care comandă intrarea $\overline{M1}$ a lui PIO, împreună cu semnalul $\overline{M1}$ generat de UC-Z80 prin intermediul unei porți 7408 ca în figura 2.35.

PIO-Z80 rămâne în starea inițială până la recepționarea unui cuvânt de comandă de la UC.

2.2.4.5. Programarea circuitului pentru funcționare în modul 0

În exemplele ce urmează se va presupune că circuitul PIO este selectat cu ajutorul bitului de adresă A_3, iar liniile de comandă C/D Sel și B/A Sel sunt acționate cu biții cei mai puțin semnificativi ai magistralei de adresă; adresele de I/E utilizate sunt, în acest caz, cele date în tabelul de mai jos:

Conexiuni externe			Adrese de I/E	Semnificația octetului transferat
$\overline{CE}(\overline{A_3})$	C/D Sel (A_1)	B/A Sel (A_0)		
0	0	0	08H	Date *port* A
0	0	1	09H	Date *port* B
0	1	0	0AH	Comenzi *port* A
0	1	1	0BH	Comenzi *port* B

Pentru a folosi unul din *port*-urile circuitului PIO-Z80, de exemplu A, în modul 0, ieşire-octet, fără a utiliza liniile de comandă a transferului şi întreruperea, este suficientă transmiterea către *port*-ul respectiv a modului şi apoi, a datelor. O secvenţă de program care realizează aceste lucruri este şi cea dată în continuare:

```
LD A,0FH            : Încărcare A cu cuvântul de comandă - mod 0
OUT (0AH),A         : Transmitere comenzi port A
LD A,77H            : Încărcare A cu octet-date
OUT (08H),A         : Transmitere date port A
```

Dacă se doreşte ca transferul de date să fie sincronizat cu echipamentul periferic, vor trebui folosite liniile de comandă şi întrerupere. În această situaţie, prin program, se vor transmite vectorul de întrerupere, modul, cuvântul de comandă-întrerupere şi se va iniţializa semnalul READY; datele vor fi trimise în subrutina de serviciu a întreruperii:

```
MOD0: IM 2             : Programare Z80 în întreruperi, modul 2
      LD HL, TAB       : În HL adresa de început a tabelei locaţiilor unde se
      LD A,H           : găsesc adresele subrutinelor de serviciu
      LD I,A           : În I octetul mai semnificativ al adresei locaţiei
      LD IX,SRVM0      : Încărcare în tabelă a adresei subrutinei
      LD (TAB+06H),IX  : de serviciu pentru PIO - modul 0
      LD A,06H         : Scriere vector de întrerupere
      OUT (0AH),A
      LD A,0FH         : Scriere cuvânt de comandă - modul 0
      OUT (0AH),A
      LD A,87H         : Scriere cuvânt de comandă-validare întreruperi
      OUT (0AH),A
      LD A,0FFH        : Iniţializare semnal ARDY printr-o scriere falsă
      OUT (08H),A      : a unui octet de date
      EI               : Validare întreruperi Z80
      ...
SRVM0:...              : Subrutina de serviciu pentru PIO - modul 0
      PUSH AF          : Salvare stare UC
      ...
      LD A,(DATE)      : Scriere octet-date
      OUT (08H),A
      ...
      POP AF           : Refacere stare UC
      ...
      EI               : Validare întreruperi
      RETI             : Revenire din subrutina de serviciu
```

2.2.4.6. Programarea circuitului pentru funcţionare în modul 1

Dacă se doreşte utilizarea unuia din *port*-urile circuitului PIO-Z80, în exemplul care urmează *port*-ul B, în modul 1, intrare-octet, prin program se vor trimite: vectorul de întrerupere, modul de lucru şi cuvântul de comandă pentru validarea întreruperilor; datele sunt preluate în subrutina de serviciu. Dăm în continuare o secvenţă de programare a *port*-ului B în modul 1:

```
MOD1: IM 2              ; Programare Z80 în întreruperi, modul 2
      LD HL,TAB         ; În HL adresa de început a tabelei locaţiilor
      LD A,H            ; unde se găsesc adresele subrutinelor de serviciu
      LD I,A            ; În I octetul mai semnificativ al adresei locaţiei
      LD IX,SRVM1       ; Încărcare în tabelă a adresei subrutinei
      LD (TAB+08H),IX   ; de serviciu pentru PIO - modul 1
      LD A,08H          ; Scriere vector de întrerupere
      OUT (0BH),A
      LD A,4FH          ; Scriere cuvânt de comandă - modul 1
      OUT (0BH),A
      LD A,87H          ; Scriere cuvânt de comandă - validare întreruperi
      OUT (0BH),A
      IN A,(09H)        ; Iniţializare BRDY printr-o citire falsă a unui octet
                        ; de date
      EI                ; Validare întreruperi Z80
      ...
SRVM1:...               ; Subrutina de serviciu pentru PIO - modul 1
      PUSH AF           ; Salvare stare UC
      ...
      IN A,(09H)        ; Citire octet-date
      ...
      POP AF            ; Refacere stare UC
      ...
      EI                ; Validare întreruperi
      RETI              ; Revenire din subrutina de serviciu
```

2.2.4.7. **Programarea circuitului pentru funcţionarea în modul 2**

Modul 2, bidirecţional, pe octet, poate fi programat numai pentru *port*-ul A, utilizând semnalele de comandă ale ambelor *port*-uri. Semnalele de comandă ale *port*-ului A sunt folosite în operaţiile de ieşire-date, iar cele ale *port*-ului B în intrări-date. Pentru ca *port*-ul A să lucreze în modul 2, *port*-ul B va trebui programat în modul 3. În continuare se dă o secvenţă de programare a *port*-ului A în modul 2.

```
MOD2: IM 2              ; Programare Z80 în întreruperi, modul 2
      LD HL,TAB         ; În HL adresa de început a tabelei locaţiilor unde se
      LD A,H            ; găsesc adresele subrutinelor de serviciu
      LD I,A            ; În I octetul mai semnificativ al adresei locaţiei
      LD IX,SRVM0       ; Încărcare adresă subrutină serviciu
      LD (TAB+06H),IX   ; ieşire-date (aceeaşi ca la modul 0)
      LD IX,SRVM1       ; Încărcare adresă subrutină serviciu
      LD (TAB+08H),IX   ; intrare-date (aceeaşi ca la modul 1)
      LD A,06H          ; Scriere vector-întrerupere pentru port-ul A
      OUT (0AH),A
      LD A,08H          ; Scriere vector-întrerupere pentru port-ul B
      OUT (0BH),A
      LD A,8FH          ; Comandă mod 2 pentru port-ul A
      OUT (0AH),A
      LD A,0CFH         ; Comandă mod 3 pentru port-ul B
      OUT (0BH),A
      LD A,0FFH         ; Liniile port-ului B definite ca intrări
      OUT (0BH),A       ; (după iniţializare toţi biţii sunt mascaţi)
      LD A,87H          ; Validare întreruperi
      OUT (0AH),A       ; pentru port-ul A
      OUT (0BH),A       ; pentru port-ul B
      LD A,0FFH         ; Iniţializare ARDY
```

```
        OUT (08H),A
        IN A,(09H)           ; Iniţializare BRDY
        EI                   ; Validare întreruperi
        ...
```

2.2.4.8. **Programarea circuitului pentru funcţionare în modul 3**

În modul 3 pot fi programate să lucreze ambele *port*-uri ale unui circuit PIO-Z80. Acest mod de funcţionare, numit şi mod de comandă, realizează operaţii de intrare-ieşire pe bit, fără să utilizeze liniile READY şi $\overline{\text{STROBE}}$. Secvenţa care urmează programează *port*-ul B în modul 3.

```
MOD3:   IM 2                 ; Programare Z80 în întreruperi, modul 2
        LD HL,TAB            ; În HL adresa de început a tabelei locaţiilor unde se
        LD A,H               ; găsesc adresele subrutinelor de serviciu
        LD I,A               ; În I octetul mai semnificativ al adresei locaţiei
        LD IX,SRVM3          ; Încărcare adresă subrutină serviciu (mod 3)
        LD (TAB+0CH),IX
        LD A,0CH             ; Scriere vector-întrerupere pentru port-ul B
        OUT (0BH),A
        LD A,0CFH            ; Comandă mod 3, pentru port-ul B
        OUT (0BH),A
        LD A,0FH             ; Definire linii de intrare pentru port-ul B
        OUT (0BH),A
        LD A,97H             ; Scriere cuvânt de comandă întreruperi
        OUT (0BH),A
        LD A,0FCH            ; Mascare toate liniile în afară de B0 şi B1
        OUT (0BH),A
        LD A,00H             ; Iniţializare linii de ieşire pe zero
        OUT (09H),A
        EI                   ; Validare întreruperi
        ...
SRVM3:  ...                  ; Subrutina de serviciu pentru modul 3
        PUSH AF              ; Salvare stare UC
        ...
        IN A,(09H)           ; Citire linii intrare port B
        AND 0FH
        LD (DISP),A          ; Afişare linii de intrare
        CALL AFIS
        RLA                  ; Transmitere linii de intrare pe liniile de ieşire
        RLA                  ; ale port-ului B
        RLA
        RLA
        OUT (09H),A
        ...
        POP AF               ; Refacere stare UC
        ...
        EI                   ; Validare întreruperi
        RETI                 ; Revenire din subrutina de serviciu
```

Modul 3 oferă o mare flexibilitate în definirea funcţionării unui *port*. De aceea mai multe opţiuni pot fi selectate şi specificate prin intermediul octeţilor de comandă.

În exemplul de mai sus, prin program se execută următoarele operaţii:

– programarea UC-Z80 în modul 2 de lucru cu întreruperile;
– iniţializarea registrului I;

– încărcarea adresei subrutinei de serviciu specifice în tabela vectorilor de întrerupere;

– scrierea vectorului de întrerupere pentru *port*-ul B;

– programarea *port*-ului B în modul 3;

– definirea registrului de selecţie I/E: 1 - linie de intrare, 0 - linie de ieşire;

– scrierea cuvântului de comandă-întreruperi: validare-întreruperi, selecţia funcţiei SAU pentru generarea întreruperii (activarea cel puţin a unei linii nemascate a *port*-ului conduce la generarea întreruperii), selecţia nivelului activ "0", specificarea faptului că urmează un cuvânt de măşti;

– transmiterea cuvântului de măşti: aici *port*-ul B este programat să genereze o întrerupere numai dacă unul din biţii B_0, B_1 devine "0".

Subrutina de serviciu SRVM3, scrisă ca exemplu, realizează următoarele funcţiuni:

– salvează starea UC;

– citeşte cele patru linii de intrare ale *port*-ului B;

– afişează aceste linii asamblate într-un octet a cărui parte semnificativă este zero (scriere la locaţia DISP, apel subrutina AFIS);

– scrie în biţii de ieşire ai *port*-ului B valoarea citită mai sus a liniilor de intrare ale aceluiaşi *port*;

– reface starea UC;

– validează întreruperile mascabile;

– redă comanda programului principal.

2.3. CIRCUITUL NUMĂRĂTOR TEMPORIZATOR CTC-Z80

Circuitul numărător-temporizator CTC-Z80 face parte din familia Z80 şi este destinat implementării funcţiilor de numărare şi măsurare a timpului. CTC poate realiza aceste funcţii pe patru canale independente de 8 biţi, interfaţându-se direct cu magistrala de date şi comenzi ale microprocesorului Z80. Circuitul poate fi programat software, astfel încât fiecare canal să lucreze independent într-unul din cele două moduri: ca numărător sau ca temporizator. În *modul numărător* CTC-Z80 numără impulsuri din exteriorul sistemului şi, dacă a fost programat să lucreze cu întreruperile, generează o întrerupere spre UC după un număr prestabilit de impulsuri. Ca şi PIO-Z80, CTC funcţionează în întreruperi conform modului 2 de lucru al UC-Z80. Vectorul şi numărul de impulsuri ce trebuie numărate până la generarea întreruperii se pot transmite software. În *modul temporizator* CTC numără impulsurile ceasului de sistem, Φ. Generând, ca şi în celălalt mod, o întrerupere după un număr prestabilit de impulsuri, circuitul asigură măsurarea precisă a unor intervale de timp definite, măsurare necesară, de exemplu, în prelucrările în timp real.

Circuitul poate fi programat uşor prin transmiterea a doi octeţi pentru fiecare canal; validarea întreruperilor se face cu ajutorul unui al treilea. Odată pornit, canalul decrementează, îşi reîncarcă la sfârşitul numărării, în mod

automat, constanta de timp şi îşi reia numărarea. Numărarea ceasului Φ sau a evenimentelor externe se face pe front pozitiv sau negativ, ales prin software. Programarea întreruperilor este simplificată, fiind necesară transmiterea unui singur octet, circuitul generând un vector unic pentru fiecare canal prin modificarea corespunzătoare a biţilor 1 şi 2.

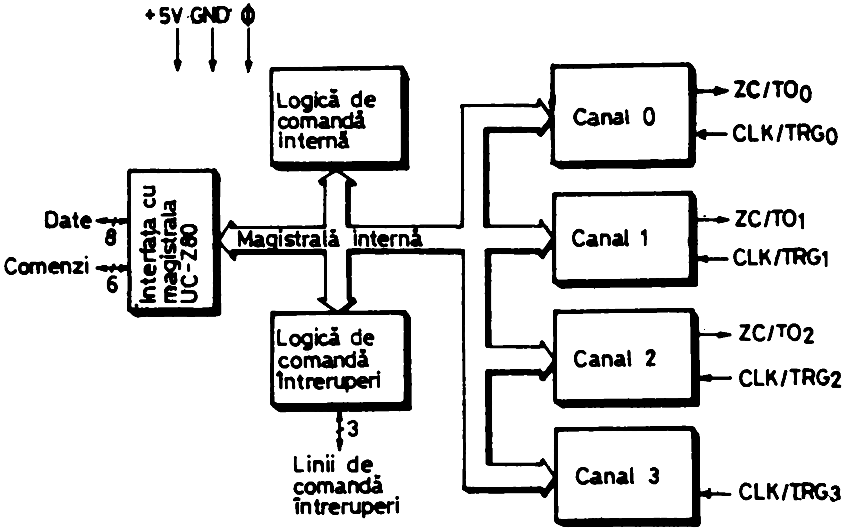


Fig. 2.36. Schema-bloc a circuitului CTC-Z80

2.3.1. STRUCTURA CIRCUITULUI CTC-Z80

În figura 2.36 se prezintă structura circuitului CTC, la nivel de schemă bloc. După cum se vede, el este organizat în jurul unei magistrale interne, fiind compus dintr-un bloc de logică pentru interfaţa cu UC-Z80, un bloc destinat funcţiei globale de comandă internă, logica de comandă a întreruperilor, cele patru canale pentru numărare sau măsurarea timpului.

Blocul de logică pentru interfaţarea cu UC-Z80 decodifică intrările de adresă şi distribuie semnalele de interfaţă pe magistrala internă a circuitului. Logica de comandă internă controlează funcţionarea globală a circuitului prin semnale generale

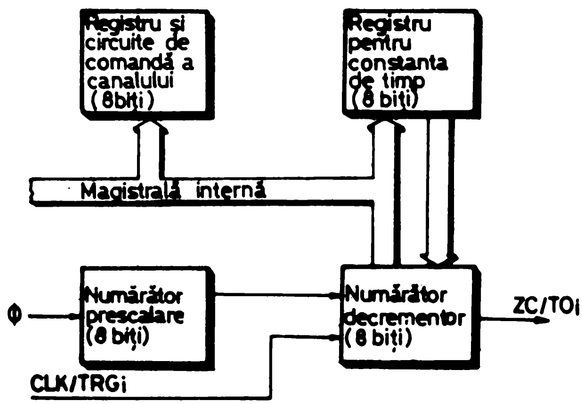


Fig. 2.37. Organizarea unui canal de numărare/măsurare a timpului

care afectează operaţii cum sunt: selecţia circuitului, iniţializarea sau logica de scriere/citire. Logica de comandă a întreruperilor asigură respectarea procedurii de cerere-achitare întrerupere conform modului 2 de lucru al microprocesorului Z80. De asemenea, această parte a circuitului controlează funcţionarea într-un lanţ de priorităţi, cu ajutorul semnalelor IEI şi IEO. Organizarea unui canal de numărare/măsurare a timpului este dată în figura 2.37. El este alcătuit din două registre, două numărătoare şi logica de comandă specifică. La programare, registrul de comandă se încarcă cu un cuvânt de comandă, logica asociată stabilind apoi funcţionarea în detaliu a canalului respectiv prin decodificarea acestui cuvânt şi precizarea următoarelor condiţii:

- validare/invalidare întreruperi;
- funcţionare în modul numărător sau în modul temporizator;
- factor de prescalare la măsurarea timpului (16 sau 256);
- frontul activ al intrării CLK/TRG;
- declanşare automată sau cu ajutorul intrării CLK/TRG în modul temporizator;
- urmează transmiterea constantei de timp;
- iniţializare software.

Registrul pentru constanta de timp memorează o valoare de 8 biţi (între 1 şi 256, 0 – reprezentând 256) ce va fi încărcată automat în numărătorul-decrementor la iniţializare şi, apoi, la fiecare trecere prin zero.

Numărătorul de prescalare, utilizat numai în modul de măsurare a timpului, divide frecvenţa ceasului de sistem cu 16 sau 256. Ieşirea acestui numărător constituie intrarea de ceas pentru numărătorul-decrementor. Acest din urmă numărător poate fi deci decrementat, fie de ieşirea numărătorului de prescalare, în modul de măsurare a timpului, fie de intrarea CLK/TRG, în modul de numărare.

UC-Z80 poate citi în orice moment, fără a perturba funcţionarea circuitului, prin execuţia unei instrucţiuni de I/E, numărul rămas de numărat până la zero. La trecerea prin zero ieşirea ZC/TO va fi pulsată şi, dacă întreruperile sunt validate, se va face o cerere de întrerupere la unitatea centrală.

După punerea sub tensiune CTC-Z80 rămâne într-o stare nedefinită. Activarea intrării $\overline{RESET}$ va conduce la iniţializarea circuitului dar, pentru a lansa numărarea sau măsurarea timpului programatorul va trebui să scrie, pentru canalul respectiv, cuvântul de comandă, constanta de timp şi, dacă se va lucra cu întreruperi, vectorul de întrerupere.

2.3.2. DESCRIEREA CONEXIUNILOR EXTERNE

CTC-Z80 are 28 de conexiuni externe, grupate, după cum se poate vedea în figura 2.38, în magistrala de date, semnale de comandă UC, semnale de comandă întreruperi şi semnale de I/E pe canale.

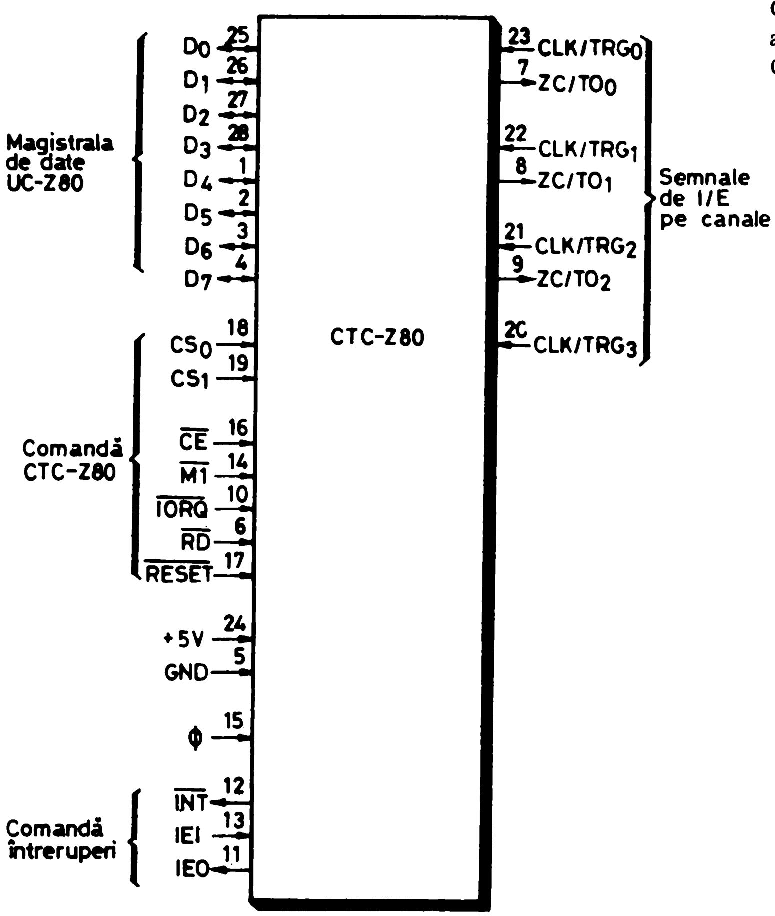


Fig. 2.38
Conexiunile externe ale circuitului CTC-Z80

2.3.2.1. **Magistrala de date**

$D_0 \div D_7$, *Data Bus*. Intrări-ieşiri trei-stări active pe "1". Magistrala se utilizează pentru transferul datelor şi comenzilor între UC şi CTC-Z80.

2.3.2.2. **Semnalele de comandă UC**

CS_0, CS_1, *Channel Select*, selecţie canal. Intrări active pe "1", folosite pentru selecţia unuia din cele patru canale independente ale circuitului. De obicei se conectează la biţii A_0 şi A_1 ai magistralei de adrese a sistemului. Selecţia canalului e conformă cu tabelul:

CS_1	CS_0	Canal selectat
0	0	Canal 0
0	1	Canal 1
1	0	Canal 2
1	1	Canal 3

$\overline{CE}$, *Chip Enable*, validare capsulă. Intrare activă pe "0". După activarea acestei intrări, CTC acceptă cuvinte de comandă, vector de întrerupere sau cuvinte cu constanta de timp, transmise pe magistrala de date, în timpul unor cicli de scriere I/E. De asemenea, dacă $\overline{CE}=0$, UC poate citi conţinutul numărătorului-decrementor printr-o instrucţiune de I/E.

$\overline{M1}$, *Machine Cycle One*, primul ciclu-maşină. Intrare de la UC-Z80, activă pe "0". Împreună cu $\overline{RD}$ semnifică un ciclu de extragere iar împreună cu $\overline{IORQ}$ un ciclu de achitare a întreruperii.

$\overline{IORQ}$, *Input/Output Request*, cerere de I/E. Intrare de la UC-Z80, activă pe "0". În timpul unui ciclu de scriere, $\overline{IORQ}$ şi $\overline{CE}$ sunt active, iar $\overline{RD}$ inactivă: CTC-Z80, ca şi PIO-Z80, nu primeşte un semnal de comandă-scriere separat, acesta generându-se intern prin inversarea lui $\overline{RD}$. La un ciclu de citire, $\overline{IORQ}$, $\overline{CE}$ şi $\overline{RD}$ sunt active, conţinutul numărătorului-decrementor fiind plasat pe magistrala de date către UC. $\overline{IORQ}$ şi $\overline{M1}$ active împreună se utilizează în ciclii de achitare a întreruperii, când canalul prioritar îşi plasează vectorul de întrerupere pe $D_0 \div D_7$.

$\overline{RD}$, *Read Cycle Status*, ciclu de citire. Intrare de la UC-Z80 activă pe "0". Semnalul utilizat împreună cu $\overline{IORQ}$ şi $\overline{CE}$ în operaţiile de transfer de date şi comenzi între UC şi CTC.

2.3.2.3. Semnalele de comandă a întreruperilor

IEI, *Interrupt Enable In*, intrare validare întreruperi. Intrare activă pe "1"; activă semnifică inexistenţa unor cereri de întrerupere prioritare în curs de tratare de către UC-Z80.

IEO, *Interrupt Enable Out*, ieşire validare întreruperi. Ieşire activă pe "1" numai când IEI=1 şi UC-Z80 nu tratează nici o întrerupere generată de vreunul din canalele de numărare/măsurare ale circuitului.

$\overline{INT}$, *Interrupt Request*, cerere întrerupere. Ieşire de tip drenă-în-gol activă pe "0". Se activează când oricare din canalele CTC căruia i s-a validat prin program generarea întreruperii trece prin zero.

$\overline{RESET}$, iniţializare. Intrare activă pe "0" utilizată pentru iniţializarea hardware a circuitului. Acest semnal opreşte numărarea în toate canalele, invalidează generarea întreruperilor, trece ieşirile ZC/TO şi $\overline{INT}$ în starea inactivă,

face IEO=IEI şi trece circuitele de comandă a magistralei de date în starea de impedanţă mare.

2.3.2.4. Semnalele de I/E pe canale

CLK/TRG$_0$ ÷ CLK/TRG$_3$, *External Clock/Timer Triger*, ceas extern/declanşare externă. Intrări active pe front pozitiv sau negativ, la alegere prin software de către utilizator. Cele patru conexiuni externe corespund celor patru canale ale CTC. În modul numărare fiecare front activ al acestor intrări decrementează numărătorul-decrementor. În modul de măsurare a timpului frontul activ este utilizat pentru începerea măsurării.

ZC/TO$_0$ ÷ ZC/TO$_2$, *Zero Count/Timeout*, trecere prin zero/terminare temporizare. Ieşire activă pe "1". Cele trei conexiuni corespund canalelor 0-2, canalul al treilea neavând o astfel de ieşire datorită limitelor impuse de capsula cu 28 de conexiuni externe. În ambele moduri la aceste ieşiri sunt generate impulsuri pozitive când numărătoarele-decrementoare trec prin zero.

2.3.3. DIAGRAME DE TIMP

2.3.3.1. Ciclul de scriere UC

Cuvântul de comandă, vectorul de întrerupere sau constantă de timp se transmit conform diagramei de timp din figura 2.39. CTC-Z80 nu are o intrare separată pentru comanda de scriere: acest semnal este generat intern dacă intrarea $\overline{RD}$ este "1" pe durata stării T$_1$. În timpul stării T$_2$ $\overline{IORQ}$ şi $\overline{CE}$ sunt pe "0" iar $\overline{M1}$ pe "1" pentru a distinge ciclul de scriere de ciclul de achitare a întreruperii. Valoarea intrărilor CS$_0$ şi CS$_1$ selectează canalul unde se vor scrie datele prezente pe magistrală, cu frontul pozitiv al ceasului din starea T$_3$.

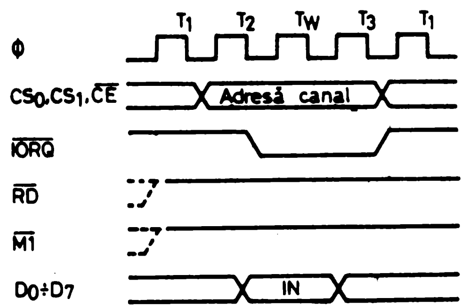


Fig. 2.39. Ciclul de scriere UC în CTC-Z80

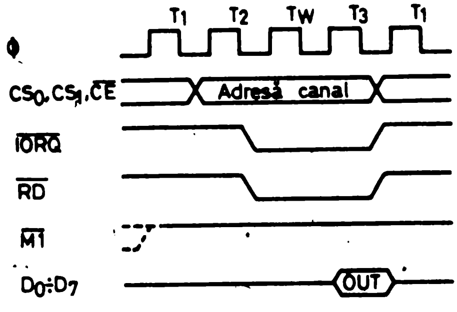


Fig. 2.40. Ciclul de citire UC din CTC-Z80

2.3.3.2. Ciclul de citire UC

Ciclul de citire, a cărui desfăşurare în timp este prezentată în figura 2.40, se utilizează pentru citirea conţinutului numărătorului-decrementor, citire efectuată fără perturbarea decrementării. În timpul stării T_2, UC-Z80 iniţiază un ciclu de citire activând intrările $\overline{RD}$, $\overline{IORQ}$ şi $\overline{CE}$ ale CTC-ului. Valoarea intrărilor CS_0 şi CS_1 selectează canalul, care va plasa, cu frontul pozitiv al ceasului din T_3, valoarea numărătorului-decrementor pe magistrala de date a sistemului.

2.3.3.3. Modul numărător

În acest mod de lucru decrementarea numărătorului-decrementor se face, aşa cum se poate vedea în figura 2.41, cu fiecare front activ al intrării CLK/TRG (aici frontul pozitiv), sincron cu ceasul Φ. Pentru ca decrementarea să apară cu primul front pozitiv al lui Φ, care urmează frontului activ al lui CLK/TRG, trebuie respectat un timp de *set-up*, de stabilizare, precizat în catalog, de exemplu 300 ns pentru versiunea CTC-Z80 fabricată de Zilog (frecvenţa ceasului Φ de 2,5 MHz). Dacă nu se respectă acest timp, numărarea va fi întârziată cu o perioadă a ceasului Φ. De asemenea, impulsul de declanşare, în figura 2.41 pozitiv, este necesar să aibă o durată minimă, iar perioada semnalului CLK/TRG să fie cel puţin de două ori mai mare decât perioada ceasului Φ. Ieşirea ZC/TO corespunzătoare va fi activată la trecerea pe zero a numărătorului-decrementor.

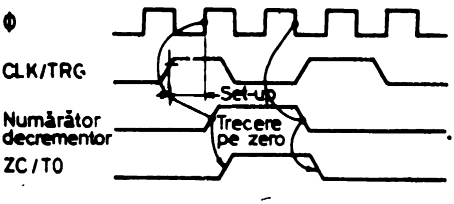


Fig. 2.41. Funcţionarea unui canal în modul numărător

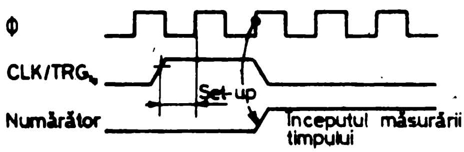


Fig. 2.42 Funcţionarea unui canal în modul temporizator

2.3.3.4. Modul temporizator

Figura 2.42 ilustrează desfăşurarea în timp a funcţionării unui canal programat să lucreze în modul temporizator. Măsurarea timpului se declanşează cu al doilea front pozitiv al ceasului Φ ce urmează frontului activ al intrării CLK/TRG, aici pozitiv. Ca şi mai sus, pentru respectarea acestei secvenţe de timp vor trebui îndeplinite condiţiile unui timp de *set-up* între frontul activ al intării CLK/TRG şi primul front pozitiv al lui Φ şi, de asemenea, cea a unei durate minime a impulsului de declanşare a măsurării timpului. Dacă frontul

activ al intrării CLK/TRG apare mai târziu decât timpul de *set-up*, inițierea numărării va fi întârziată cu o perioadă de ceas.

Precizăm că măsurarea timpului poate fi declanșată și automat, fără utilizarea conexiunii externe CLK/TRG, prin transmiterea unei anumite configurații a cuvântului de comandă.

2.3.4. PROGRAMAREA CIRCUITULUI CTC-Z80

Pentru a programa circuitul CTC-Z80 se transmit cuvântul de comandă, constanta de numărare și, dacă se dorește utilizarea întreruperilor, vectorul de întrerupere. Cuvântul de comandă este folosit pentru selectarea modului de lucru și a altor parametri ce vor caracteriza funcționarea canalului programat. Constanta de timp, o valoare binară cuprinsă în intervalul (1, 256), este întotdeauna precedată de un cuvânt de comandă. Actualizarea cuvântului de comandă și/sau a constantei de timp se poate face în orice moment pe timpul numărării, cu observația că noua constantă de timp va fi încărcată în numărătorul-decrementor ulterior, la trecerea acestuia prin zero.

Dacă cel puțin unul din canalele de numărare ale circuitului CTC-Z80 are întreruperi validate, bitul 7 al cuvântului de comandă este "1", procedura de programare va cuprinde și transmiterea vectorului de întrerupere. Este necesară scrierea unui singur vector pentru întregul circuit deoarece logica de comandă a întreruperilor modifică în mod automat vectorul plasat pe magistrală de către CTC, în ciclul de achitare, precizând canalul care solicită întreruperea.

Selecția canalului programat se face cu ajutorul intrărilor CS_0 și CS_1 conform tabelului dat în § 2.3.2.

2.3.4.1. **Cuvântul de comandă**

Pentru a încărca un cuvânt de comandă, unitatea centrală execută o instrucțiune de I/E la adresa corespunzătoare canalului ales. Octetul transmis va fi interpretat drept cuvânt de comandă și înscris în registrul de comandă al canalului respectiv dacă bitul 0, cel mai puțin semnificativ, este "1". Cu ajutorul celorlalți șapte biți se selectează modul de lucru și parametrii de funcționare. Formatul cuvântului de comandă este următorul:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Validare întreruperi	Mod de lucru	Factor prescalare	Front activ	Mod de declanșare	Urmează constanta de timp	Inițializare	1

D_7, *validare-întreruperi*. Dacă $D_7=1$, canalul respectiv este validat să genereze o întrerupere la fiecare trecere prin zero a numărătorului-decrementor. În această situaţie, se încarcă în CTC şi vectorul de întrerupere. Întreruperile pot fi validate în oricare din cele două moduri de lucru. Dacă se trimite un cuvânt de comandă de actualizare la un canal în funcţionare, cuvânt având $D_7=1$, validarea întreruperilor nu va fi retroactivă în raport cu o posibilă trecere anterioară a numărătorului prin zero. $D_7=0$ înseamnă invalidarea întreruperilor, ştergerea întreruperii în aşteptare pe canalul respectiv. Ca şi la PIO poate apărea un asincronism între generarea întreruperii de către CTC şi invalidarea ei de către UC, situaţie care, datorită citirii unui vector fals de întrerupere, ar conduce la o eroare de program. Pentru a rezolva această problemă se poate utiliza următoarea secvenţă de program:

```
...
LD A,01H         ; Încărcare A cu comanda de invalidare
DI               ; Invalidare întreruperi UC
OUT (CTC),A      ; Invalidare întreruperi CTC
EI               ; Validare întreruperi UC
...
```

D_6, *mod de lucru*. Dacă $D_6=1$ este selectat modul numărător în care numărătorul-decrementor al canalului se decrementează la fiecare front activ al intrării CLK/TRG respective. În acest mod nu se utilizează numărătorul de prescalare. Când $D_6=0$, se selectează modul temporizator în care numărătorul-decrementor este acţionat de ieşirea numărătorului de prescalare. La ieşirea ZC/TO a canalului se generează impulsuri cu o perioadă dată de:

$$T_{ZC/TO}=t_{\Phi}\cdot FP\cdot CT,$$

unde t_{Φ} este perioada ceasului de sistem, Φ, FP - factorul de prescalare, 16 sau 256, iar CT - constanta de timp încărcată în registrul pentru constanta de timp al canalului respectiv.

D_5, *factor-prescalare*. Bitul D_5, definit numai pentru modul temporizator, precizează cu cât trebuie să se dividă frecvenţa ceasului Φ: cu 16, $D_5=0$, sau cu 256, $D_5=1$.

D_4, *front activ*. Acest bit stabileşte care din fronturile impulsului CLK/TRG va fi utilizat pentru lansarea măsurării timpului în modul temporizator, respectiv pentru decrementarea în modul numărător. Frontul pozitiv este selectat cu $D_4=1$ iar cel negativ cu $D_4=0$. O reprogramare a frontului activ pe durata numărării echivalează cu un front activ, conducând la decrementarea numărătorului-decrementor. De asemenea, o schimbare a frontului activ, printr-un cuvânt de comandă de actualizare, în timp ce canalul aşteaptă să înceapă măsurarea timpului, conduce la startarea numărătoarelor, echivalând deci cu un impuls de declanşare.

D_3, *mod de declanşare*. Definit numai pentru modul temporizator, acest bit specifică modul de declanşare, de începere a măsurării timpului: $D_3=1$, declanşare externă cu ajutorul intrării CLK/TRG (vezi figura 2.42); $D_3=0$,

declanşare internă, automată, când începerea măsurării timpului se face pe frontul pozitiv al ceasului Φ în starea T_2 din ciclul-maşină următor celui în care s-a scris constanta de timp. După declanşare, numărarea se face în mod continuu, la trecerea prin zero constantele încărcându-se automat, numărătoarele fiind decrementate fără întrerupere sau întârziere până la o iniţializare hardware sau software.

D_2, *urmează constanta de timp.* $D_2=1$ indică faptul că următorul cuvânt scris la adresa canalului programat este o constantă de timp, ce poate fi transmisă în orice moment, numărătorul-decrementor continuând decrementarea constantei anterioare până la trecerea prin zero, după care o va încărca pe cea nouă. $D_2=0$ înseamnă că nu urmează o constantă. Cuvântul de comandă transmis cu $D_2=0$ este de obicei un cuvânt de actualizare a registrului de comandă asociat canalului deja în funcţiune. Aceasta deoarece un canal nu poate funcţiona fără să i se fi transmis în mod corect constanta de timp şi singurul mod de a o transmite este prin poziţionarea pe "1" a bitului D_2, într-un cuvânt de comandă anterior.

D_1, *iniţializare.* Poziţionarea pe "1" a acestui bit conduce la o iniţializare software, canalul oprindu-se din numărare sau din măsurarea timpului. Scrierea în D_1 a unui impuls de iniţializare opreşte funcţionarea canalului, dar nu modifică nici unul din biţii registrului de comandă. Dacă $D_2=D_1=1$, canalul îşi va relua funcţionarea după încărcarea constantei de timp. Dacă $D_1=0$, canalul îşi continuă funcţionarea curentă.

2.3.4.2. **Constanta de timp**

Pentru a începe numărarea în oricare din cele două moduri de lucru, un canal CTC trebuie să primească o constantă de timp. Constanta de timp poate avea orice valoare întreagă cuprinsă între 1 şi 256, cu 00H interpretat ca 256. Încărcarea unei constante se face în doi paşi: întâi transmiterea unui cuvânt de comandă cu bitul $D_2=1$, apoi scrierea unui octet având valoarea constantei. Transmiterea unei constante spre un canal aflat în funcţiune va conduce la încărcarea ei în registrul pentru constantă, de unde va fi transferată în numărătorul-decrementor, la terminarea operaţiei curente, adică la trecerea acestuia prin zero.

2.3.4.3. **Vectorul de întrerupere**

Vectorul de întrerupere se transmite dacă cel puţin unul din canalele CTC are întreruperile validate. CTC-Z80 lucrează în întreruperi la fel ca PIO-Z80, respectând procedura corespunzătoare modului 2 al microprocesorului Z80 (vezi § 2.1, 2.2). Programarea vectorului constă în scrierea unui octet în canalul 0

al CTC. Acest octet trebuie să aibă bitul cel mai puţin semnificativ $D_0=0$, pentru a fi distins de un cuvânt de comandă. Utilizatorul precizează în acest octet cei mai semnificativi cinci biţi, logica de comandă din CTC urmând a stabili valorile biţilor D_2 şi D_1 în funcţie de canalul care solicită întreruperea. Formatul vectorului de întrerupere este deci următorul:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	X	X	0

D_7-D_3 se precizează de către utilizator iar D_2-D_1 reprezintă un identificator de canal generat de CTC-Z80: 00 = canal 0, 01 = canal 1, 10 = canal 2 şi 11 = canal 3.

Canalul 0 este prioritar faţă de celelalte canale.

2.3.4.4. Programarea circuitului pentru funcţionare în modul numărător

În exemplele următoare se va considera că circuitul CTC-Z80 este selectat cu ajutorul bitului de adresă A_4, iar liniile CS_0, CS_1 sunt comandate de biţii cei mai puţin semnificativi ai magistralei de adrese; în consecinţă adresele de I/E corespunzătoare celor patru canale vor fi cele din tabelul de mai jos:

Conexiuni externe			Adrese de I/E	Canal selectat
$\overline{CE}(\overline{A_4})$	$CS_1(A_1)$	$CS_0(A_0)$		
0	0	0	10H	Canal 0
0	0	1	11H	Canal 1
0	1	0	12H	Canal 2
0	1	1	13H	Canal 3

Canalul programat să lucreze în modul numărător îşi decrementează numărătorul-decrementor la fiecare front activ al intrării externe CLK/TRG. Dacă întreruperile sunt validate, la trecerea prin zero va fi generată o întrerupere, decrementarea reluându-se după încărcarea constantei păstrate în

registrul pentru constanta de timp. Se dă în continuare o secvență de programare a canalului 2 în modul numărător:

```
MDNUM:IM 2              ; Programarea Z80 în întreruperi, modul 2
      LD HL,TAB         ; În HL adresa de început a tabelei locaţiilor unde se
      LD A,H            ; găsesc adresele subrutinelor de serviciu
      LD I,A            ; În I octetul mai semnificativ al adresei locaţiei
      LD IX,SRVC2       ; Încărcare în tabelă a adresei subrutinei de serviciu
      LD (TAB+1AH),IX   ; pentru CTC - canalul 2, modul numărător
      LD A,18H          ; Scriere vector de întrerupere la canalul 0
      OUT (10H),A
      LD A,0C7H         ; Scriere cuvânt de comandă canal 2
      OUT (12H),A
      LD A,(CONST)      ; Scriere constantă de timp canal 2
      OUT (12H),A
      EI                ; Validare întreruperi
      ...
```

Secvenţa MDNUM programează canalul 2 al CTC pentru a funcţiona cu întreruperile validate. În acest scop secvenţa trebuie să precizeze coţinutul a trei din registrele circuitului: registrul pentru vectorul de întrerupere, registrul de comandă al canalului 2 şi registrul pentru constanta de timp, de asemenea al canalului 2.

Vectorul de întrerupere, scris întotdeauna la adresa de I/E a canalului 0, are cei mai semnificativi cinci biţi poziţionaţi prin program, ceilalţi trei biţi fiind determinaţi de CTC. Semnificaţia cuvântului de comandă, 0C7H, recunoscut datorită faptului că are $D_0=1$, este următoarea:

$D_7=1$ validează întreruperile generate de canalele CTC. În cazul nostru trecerea prin zero a numărătorului-decrementor din canalul 2 va conduce la activarea liniei $\overline{INT}$;

$D_6=1$ programează canalul 2 să lucreze în modul numărător;

$D_5=0$ neutilizat în modul numărător;

$D_4=0$ specifică frontul negativ al intrării CLK/TRG_2, ca front activ pe care se decrementează numărătorul;

$D_3=0$ neutilizat în modul numărător;

$D_2=1$ înseamnă că urmează scrierea constantei de timp;

$D_1=1$ specifică faptul că CTC - canalul 2 va începe să funcţioneze, să numere, după încărcarea constantei de timp.

După execuţia secvenţei MDNUM, numărătorul-decrementor al canalului 2 va fi decrementat cu o unitate la fiecare front negativ apărut la intrarea CLK/TRG_2. După un număr de impulsuri egal cu constanta de timp, CTC va genera o întrerupere activând linia $\overline{INT}$. În acelaşi timp, constanta va fi reîncărcată în numărătorul-decrementor contorizându-se în continuare fronturile negative ale semnalului CLK/TRG_2. Dacă subrutina de serviciu a întreruperii durează mai mult decât decrementarea constantei de timp, se pot pierde întreruperi, deoarece circuitul CTC-Z80 nu memorează o întrerupere în aşteptare pe timpul cât întreruperea precedentă, generată de acelaşi canal, se află în curs de tratare. În această situaţie este asigurată numai corectitudinea funcţiei de numărare.

2.3.4.5. **Programarea circuitului pentru funcţionare în modul temporizator**

Vom ilustra programarea circuitului CTC-Z80 în modul temporizator printr-un exemplu de realizare a unui cronometru. Pentru aceasta, canalele 0, 1, 2 se conectează în cascadă, ieşirile ZC/TO_0 şi ZC/TO_1 legându-se la intrările CLK/TRG_1, respectiv CLK/TRG_2. Canalul 0 va lucra în modul temporizator iar canalele 1 şi 2 în modul numărător. Pornirea externă a cronometrului, a măsurării timpului, se face cu ajutorul unui impuls la intrarea CLK/TRG_0.

Numărul de secunde trecute, maximum 255, se înregistrează în numărătorul-decrementor al canalului 2. Pentru citirea numărului de secunde, cu ajutorul unei subrutine de serviciu, se utilizează canalul 3, programat în modul numărător. Acesta va genera o întrerupere la orice cerere externă de sfârşit al cronometrării, cerere materializată printr-un impuls la intrarea CLK/TRG_3. Secvenţa de programare a celor patru canale ale CTC se dă mai jos:

```
MODT: IM 2                ; Programare Z80 în întreruperi modul 2
      LD HL,TAB           ; În HL adresa de început a tabelei locaţiilor
      LD A,H              ; unde se găsesc adresele subrutinelor de serviciu
      LD I,A              ; În I octetul mai semnificativ al adresei locaţiei
      LD IX,SRVCR         ; Încărcare în tabelă a adresei subrutinei
      LD (TAB+26H),IX     ; de serviciu pentru CTC - cronometru
                          ; (vector generat de canalul 3)
      LD A,26H            ; Scriere vector întrerupere la canalul 0
      OUT (10H),A
      LD A,2FH            ; Cuvânt de comandă canal 0
      OUT (10H),A
      LD A,98H            ; Constanta de timp pentru canalul 0
      OUT (10H),A
      LD A,47H            ; Cuvânt de comandă canal 1
      OUT (11H),A
      LD A,40H            ; Constanta de timp pentru canalul 1
      OUT (11H),A
      LD A,47H            ; Cuvânt de comandă canal 2
      OUT (12H),A
      LD A,00H            ; Constanta de timp pentru canalul 2
      OUT (12H),A
      LD A,0C7H           ; Cuvânt de comandă canal 3
      OUT (13H),A
      LD A,01H            ; Constanta de timp pentru canalul 3
      OUT (13H),A
      EI                  ; Validare întreruperi
      ...
SRVCR:...                 ; Subrutina de serviciu pentru CTC-cronometru
      LD C,12H            ; Citire canal 2 (număr secunde trecute)
      IN B,(C)
      XOR A               ; Ştergere A
      SUB B               ; În A numărul de secunde trecute
      LD (DATE),A         ; Afişarea numărului de secunde
      CALL AFIS
      LD A,2FH            ; Reiniţializare cronometru
      OUT (10H),A         ; Cuvânt de comandă canal 0
      LD A,98H            ; Constanta de timp pentru canalul 0
      OUT (10H),A
      LD A,47H            ; Cuvânt de comandă canal 1
      OUT (11H),A
```

```
LD A,40H        ; Constanta de timp pentru canalul 1
OUT (11H),A
LD A,47H        ; Cuvânt de comandă canal 2
OUT (12H),A
LD A,00H        ; Constanta de timp pentru canalul 2
OUT (12H),A
LD A,0C7H       ; Cuvânt de comandă canal 3
OUT (13H),A
LD A,01H        ; Constanta de timp pentru canalul 3
OUT (13H),A
...
EI              ; Validare întreruperi
RETI            ; Revenire din subrutina de serviciu
```

Secvenţa MODT programează canalul 0 în modul temporizator şi canalele 1, 2, 3 în modul numărător. Întreruperile sunt validate numai pentru canalul 3. Ca şi în cazul secvenţei anterioare de programare, se va preciza întâi conţinutul registrului vector de întrerupere şi, apoi, pentru fiecare canal în parte, al registrului de comandă şi al registrului constantă de timp. Cuvântul de comandă 2FH, transmis canalului 0, are următoarea semnificaţie:

D_7 = 0 invalidează întreruperile pe canalul 0;

D_6 = 0 programează canalul 0 să lucreze în modul temporizator;

D_5 = 1 precizează că factorul de prescalare este 256;

D_4 = 0 specifică frontul negativ al impulsului de la intrarea CLK/-TRG_0 ca front activ de lansare a operaţiei de măsurare a timpului;

D_3 = 1 specifică faptul că lansarea operaţiei de măsurare a timpului se va face din exterior, cu ajutorul unui impuls la intrarea CLK/TRG_0;

D_2 = 1 înseamnă că urmează scrierea constantei de timp;

D_1 = 1 specifică faptul că pentru canalul 0 funcţionarea va începe sau va fi reluată după încărcarea constantei de timp.

Cuvântul de comandă 47H, transmis canalelor 1 şi 2, are aceeaşi semnificaţie ca în cazul secvenţei anterioare MDNUM, singura deosebire constând în aceea că nu validează întreruperile. Acest cuvânt de comandă programează, aşa cum am mai spus, canalele 1 şi 2 să lucreze în modul numărător, numărătoarele lor fiind decrementate la fiecare impuls generat la ieşirile ZC/TO_0, respectiv ZC/TO_1. Cuvântul de comandă 0C7H este identic cu cel din secvenţa MDNUM, el programând canalul 3, de asemenea în modul numărător, dar cu întreruperi validate. Acest din urmă canal va fi utilizat numai pentru generarea unei întreruperi în momentul încheierii operaţiei de cronometrare. De aceea, canalul se încarcă cu constanta 01H, întreruperea activându-se la terminarea măsurării timpului, când transmiterea din exterior a unui impuls la intrarea CLK/TRG_3 va conduce la decrementarea numărătorului-decrementor şi trecerea lui prin zero.

Subrutina SRVCR citeşte numărătorul-decrementor al canalului 2 în care se găseşte numărul cronometrat de secunde, de fapt restul scăderii acestui număr din constanta 00H, calculând apoi valoarea reală a perioadei de timp măsurate. În continuare se afişează timpul cronometrat şi se reiniţializează cronometrul,

transmiţând aceleaşi cuvinte de comandă şi constante de timp pentru cele patru canale ale CTC, ca şi în secvenţa MODT din programul principal.

2.4. CIRCUITUL PENTRU COMANDA INTRĂRILOR/IEŞIRILOR SERIE SIO-Z80

2.4.1. STRUCTURA CIRCUITULUI SIO-Z80

Circuitul pentru comanda intrărilor/ieşirilor serie SIO-Z80 face parte din familia de circuite Z80 şi este destinat, în principal, implementării aplicaţiilor de comunicaţii de date, asigurând interfaţarea independentă a două canale. Circuitul realizează interfaţa serială binară între echipamente de prelucrare de date şi echipamente de comunicaţii, modemuri, convertind informaţia paralel-serie şi serie-paralel şi permiţând implementarea practic a tuturor protocoalelor de comunicaţie uzuale, de tip asincron şi sincron, orientate pe caracter sau pe bit. SIO-Z80 este un dispozitiv de I/E deosebit de flexibil, programabil software, ce înglobează funcţiile unor circuite tradiţionale de tip UART, *Universal Asynchronous Receiver/Transmitter*, receptor/transmiţător asincron universal, cum sunt TMS6011, S1883, de tip USART, *Universal Synchronous/Asynchronous Receiver/Transmitter*, receptor/transmiţător asincron/sincron universal, ca, de exemplu, 8251, 8251A, 2651, şi ale unor circuite de comandă pentru comunicaţii sincrone ca 2652 sau 8273. Menţionăm posibilităţile de lucru ale circuitului cu întreruperi vectorizate sau nevectorizate, în mod DMA sau în modul de testare software, *polling*. De asemenea, SIO-Z80 poate fi folosit, cu restricţiile impuse în primul rând de viteza de transfer, pentru conectarea oricăror echipamente periferice cu interfeţe serie, de pildă unităţi de discuri flexibile, imprimante sau console.

Dintre caracteristicile principale ale circutului enumerăm:

– poate comanda independent două canale *duplex*[1];

– vitezele de transmisie 0 ÷ 500 Kbiţi/s, pentru frecvenţa ceasului de 2,5 MHz (SIO-Z80) sau 0 ÷ 800 Kbiţi/s, pentru ceas de 4 MHz (SIO-Z80A);

– *buffer* de recepţie cu patru niveluri şi *buffer* de transmisie cu două niveluri;

– mod de transmisie asincron cu 5, 6, 7 sau 8 biţi/caracter, 1, 11/2 sau 2 biţi de STOP, cu sau fără paritate, rate de transfer de x1, x16, x32 sau x64 perioada ceasului, generare şi detecţie de caractere BREAK, detecţie de erori de paritate, de depăşire sau de cadrare;

---

[1] Transmiterea datelor simultan în ambele sensuri defineşte un canal *duplex*; transmiterea datelor într-un sens sau în celălalt, nesimultan, defineşte un canal *semiduplex*, iar transmiterea numai într-un singur sens caracterizează un canal *simplex*.

– mod de transmisie sincron cu sincronizare pe caracter, internă sau externă, generarea unuia sau a două caractere de sincronizare, inserarea automată de caractere de sincronizare, generarea şi verificarea de tip CRC; permite implementarea unor protocoale cum sunt BISYNC, SDLC, HDLC asigurând inserarea delimitatorului, inserarea şi ignorarea de biţi "0", manipularea resturilor din câmpul de informaţie;

– posibilitatea de a lucra în întreruperi conectat într-un lanţ de priorităţi.

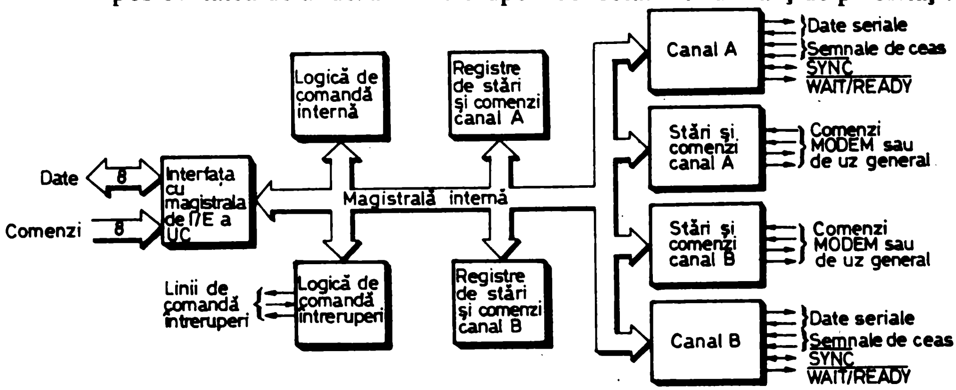


Fig. 2.43. Schema-bloc a circuitului SIO-Z80

Schema bloc a circuitului SIO-Z80 se prezintă în figura 2.43. Aşa cum se observă, circuitul este organizat pe principiul unei magistrale interne, la care sunt conectate blocurile funcţionale: interfaţa cu magistrala de I/E a unităţii centrale, logica de comandă a întreruperilor, logica de comandă internă, canalele seriale *duplex* A şi B.

Fiecare din cele două canale este constituit, la rândul lui, din registre de stări şi comenzi, logica de stări şi comenzi pentru interfaţa cu modemul sau cu alte echipamente externe, partea de transmisie/recepţie propriu-zisă. Alcătuirea acestei părţi de transmisie/recepţie pentru canalul A se dă în figura 2.44. La recepţie se observă cele patru niveluri ale *buffer*-ului, trei alcătuind o stivă de tip FIFO, *First-In First-Out*, iar al patrulea fiind reprezentat de registrul pentru conversia serie-paralel. Această memorie tampon permite o întârziere, necesară, de exemplu, pentru tratarea unei întreruperi de început de bloc în cazul unei transmisii de viteză mare. La transmisie există un registru de 8 biţi ce se încarcă de pe magistrala internă de date şi un registru de deplasare de 20 de biţi care poate fi încărcat cu date sau cu caractere SYNC.

Funcţionalitatea fiecărui canal se defineşte prin software cu ajutorul a opt registre de comandă ce pot fi programate. Aceste registre de comandă, WR0 ÷ WR7, au următoarele funcţii principale:

WR0 – adresare celelalte registre, iniţializare CRC, comenzi de iniţializare pentru diverse moduri de lucru;

WR1 – definire mod de transfer şi întreruperi transmisie/recepţie;

WR2 – vector de întrerupere (scris numai pentru canalul B);

WR3 – comenzi şi parametri recepţie;

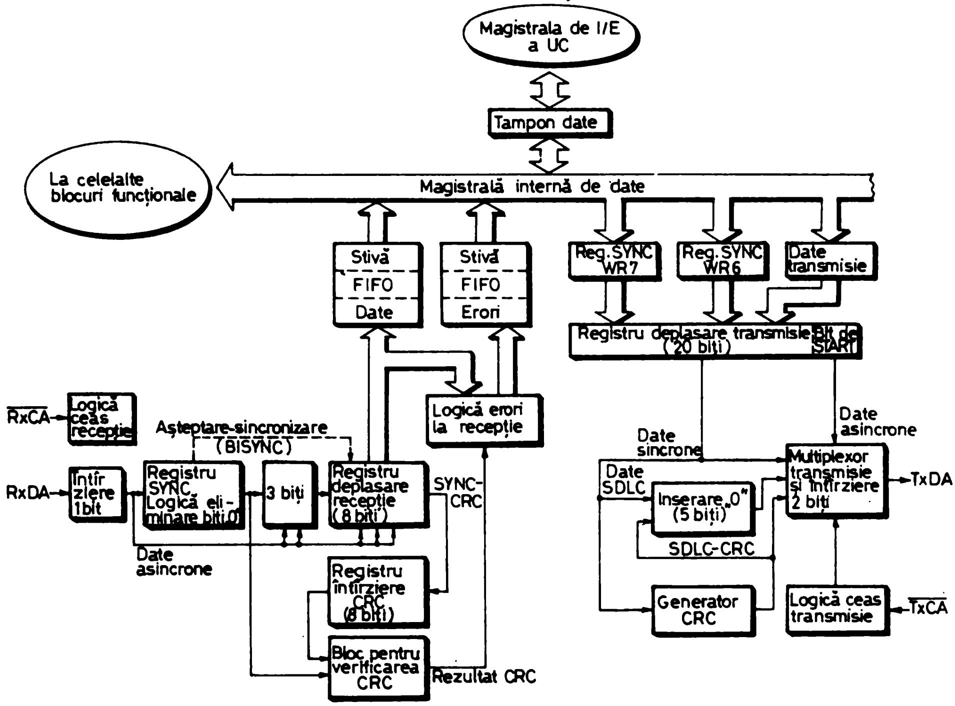


Fig. 2.44. Organizarea canalului serial A

WR4 – moduri şi parametri diverşi transmisie/recepţie;
WR5 – comenzi şi parametri transmisie;
WR6 – caracter SYNC sau câmp de adresă SDLC;
WR7 – caracter SYNC sau delimitator SDLC.

Pentru a cunoaşte starea fiecărui canal, programatorul poate citi trei registre de stare. Aceste trei registre, RR0 ÷ RR2 dau informaţii despre condiţiile de eroare, vectorul de întrerupere sau starea unor semnale de interfaţă standard:

RR0 – stare *buffer* de transmisie/recepţie, stare de întrerupere, stare externă;
RR1 – stare condiţie de recepţie specială;
RR2 – vector de întrerupere modificat (citit numai pe canalul B).

Observăm că vectorul de întrerupere al circuitului se scrie numai în registrul WR2 al canalului B şi se citeşte, modificat de starea dispozitivului, în registrul RR2, de asemenea al canalului B.

2.4.2. DESCRIEREA CONEXIUNILOR EXTERNE

Datorită constrângerilor impuse de necesitatea de a împacheta circuitul într-o capsulă cu 40 de conexiuni externe SIO-Z80 se fabrică în trei versiuni de

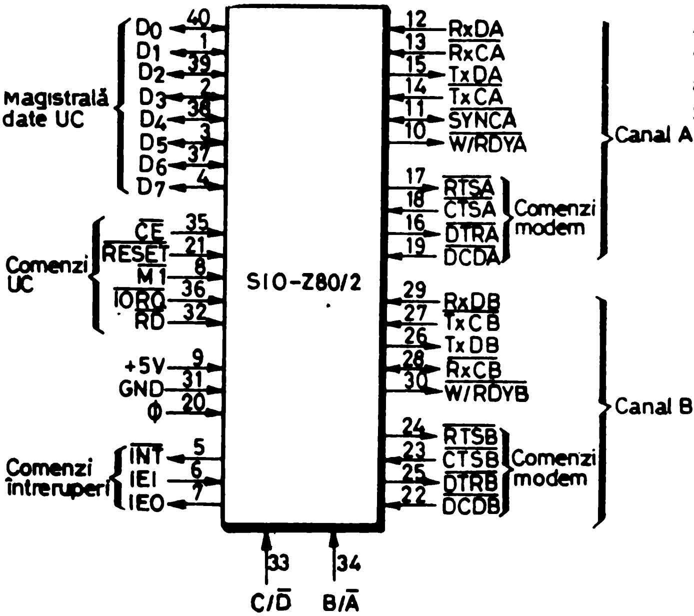


Fig. 2.45. Conexiunile externe ale circuitului SIO-Z80/2

încapsulare. Conexiunile externe ale variantei preferate în majoritatea aplicaţiilor, SIO-Z80/2, sunt date în figura 2.45. În tabelul 2.13 sunt date configuraţiile conexiunilor care diferă pentru cele trei variante: SIO-Z80/2 nu are $\overline{SYNCB}$, lui SIO-Z80/1 îi lipseşte semnalul $\overline{DTRB}$, iar la SIO-Z80/0 semnalele $\overline{TxCB}$ şi $\overline{RxCB}$ sunt legate împreună.

TABELUL 2.13. **Conexiuni externe diferite ale variantelor SIO-Z80**

Conexiunea externă	SIO-Z80/2	SIO-Z80/1	SIO-Z80/0
25	$\overline{DTRB}$	TxDB	$\overline{DTRB}$
26	TxDB	$\overline{TxCB}$	TxDB
27	$\overline{TxCB}$	$\overline{RxCB}$	$\overline{RxTxCB}$
28	$\overline{RxCB}$	RxDB	RxDB
29	RxDB	$\overline{SYNCB}$	$\overline{SYNCB}$

Vom da în continuare semnificaţia semnalelor specifice circuitului, diferite de acelea care permit conectarea lui la sisteme realizate cu microprocesorul Z80. Aceste din urmă semnale, $D_0 \div D_7$, $\overline{CE}$, $\overline{M1}$, $\overline{IORQ}$, $\overline{RD}$, Φ, $\overline{INT}$, IEI şi IEO, sunt identice cu cele descrise în § 2.2 şi 2.3.

B/$\overline{A}$, selecţie canal A sau B. Intrare cu ajutorul căreia se precizează canalul cu care UC face transferul pe timpul unei operaţii de I/E. De obicei, la

această conexiune se leagă bitul A_0 al magistralei de adrese. "1" selectează canalul B, iar "0" canalul A.

$C/\overline{D}$, selecţie date sau comenzi/stări. Intrare prin intermediul căreia se determină tipul informaţiei transferate între UC şi SIO pe timpul unei operaţii de I/E. $C/\overline{D}=1$ pe durata unei scrieri I/E va conduce la interpretarea de către SIO a informaţiei de pe magistrala de date $D_0 \div D_7$ ca o comandă, iar $C/\overline{D}=0$ va face ca informaţia să fie considerată date. O citire cu $C/\overline{D}=1$ înseamnă o citire de stare. De obicei, la această conexiune se leagă bitul A_1 al magistralei de adrese.

$\overline{RESET}$, iniţializare. Intrare activă pe "0". La activarea acestei linii se invalidează circuitele de recepţie şi de transmisie, se forţează ieşirile de emisie date $\overline{TxDA}$ şi $\overline{TxDB}$ în starea MARK, se trec pe "1" toate comenzile către modem şi se invalidează întreruperile. După o astfel de iniţializare, înainte de a recepţiona sau transmite date, este necesară reprogramarea tuturor registrelor de comenzi.

$\overline{RxDA}$, $\overline{RxDB}$, *Receive Date*, recepţie date. Intrări serie active pe "1", ce admit semnale cu niveluri TTL.

$\overline{RxCA}$, $\overline{RxCB}$, *Receiver Clock*, baze de timp pentru recepţie. Datele recepţionate se eşantionează cu frontul pozitiv al semnalelor primite la aceste intrări. Ceasurile $\overline{RxC}$ pot avea, în modurile asincrone, o frecvenţă egală, de 16, 32 sau 64 ori mai mare decât viteza de transmisie a datelor. SIO-Z80 nu are nevoie de semnale de ceas cu factor de umplere egal, ceea ce permite generarea lor cu ajutorul unui circuit CTC-Z80. Cele două intrări sunt de tip *trigger-Schmitt*, fără o margine garantată a nivelului de zgomot, admiţând fronturi pozitive şi negative lente.

$\overline{TxDA}$, $\overline{TxDB}$, *Transmit Data*, emisie date. Ieşiri serie, niveluri TTL, active pe "1". Datele generate la aceste ieşiri sunt schimbate pe fronturile negative ale ceasurilor $\overline{TxC}$.

$\overline{TxCA}$, $\overline{TxCB}$, *Transmitter Clocks*, baze de timp pentru emisie. Intrări de ceas de tip *trigger-Schmitt* care, ca şi ceasurile de recepţie, pot avea în modurile asincrone frecvenţa egală de 16, 32 sau 64 ori mai mare decât rata de transfer. Este obligatoriu ca factorul de multiplicare pentru emiţător şi receptor să fie acelaşi. De asemenea, $\overline{TxC}$ se poate obţine cu ajutorul unui CTC.

$\overline{SYNCA}$, $\overline{SYNCB}$, *Synchronization*, sincronizare. Intrări/ieşiri active pe "0". În timpul recepţiei asincrone aceste conexiuni externe sunt folosite ca intrări similare cu $\overline{CTS}$ şi $\overline{DCD}$ [27]. În acest mod, afectează biţii de stare SYNC/aşteptare-sincronizare din registrele de stare RR0, fiind utilizabile pentru orice funcţie de intrare. În plus, SIO sesizează tranziţiile de nivel şi întrerupere UC. Precizăm că, dacă se utilizează modul asincron şi întreruperile (externe/de stare) sunt validate, intrarea SYNC nu trebuie lăsată în gol pentru a nu se genera întreruperi parazite. În modul de sincronizare externă, $\overline{SYNCA}$, $\overline{SYNCB}$ sunt tot intrări comandate pe "0" cu al doilea front pozitiv al semnalului $\overline{RxC}$ după frontul pozitiv al $\overline{RxC}$ cu care s-a eşantionat ultimul bit

al caracterului de sincronizare. Cu alte cuvinte, după detecţia configuraţiei de sincronizare, logica externă trebuie să aştepte două perioade complete ale ceasului de recepţie şi apoi să activeze intrările SYNC. După trecerea pe "0" intrarea SYNC trebuie menţinută în această stare până când UC informează logica externă de detecţie a sincronizării că s-a pierdut sincronizarea sau că poate să înceapă un nou mesaj. În modul de sincronizare externă asamblarea caracterului începe cu primul front pozitiv al $\overline{RxC}$ după activarea intrării SYNC. În modul de sincronizare internă, MONOSYNC sau BYSYNC, cele două conexiuni externe sunt utilizate ca ieşiri activate pe durata unei părţi din perioada ceasului, $\overline{RxC}$, în care sunt recunoscute caracterele de sincronizare. Recunoaşterea caracterelor de sincronizare nu se memorează, astfel încât ieşirile SYNC vor fi activate de fiecare dată când se recunoaşte o configuraţie de sincronizare.

$\overline{W}/\overline{RDYA}$, $\overline{W}/\overline{RDYB}$, *Wait/Ready A, Wait/Ready B*, aşteptare/gata A, B. Ieşiri de tip drenă-în-gol, atunci când sunt programate pentru a realiza funcţia $\overline{WAIT}$, sau comandate pe "1" şi "0", când sunt programate pentru funcţia $\overline{READY}$. Se programează ca ieşiri de tip READY, pentru a comanda circuite DMA, de exemplu 8257, şi ca ieşiri de tip WAIT, pentru a sincroniza unitatea centrală la rata de transmisie a SIO. La iniţializare ieşirile se trec în starea drenă-în-gol, $\overline{WAIT}$.

$\overline{RTSA}$, $\overline{RTSB}$, *Request To Send*, cerere pentru emisie. Ieşiri active pe "0". Ieşirea trece pe "0" atunci când bitul RTS din registrul de comandă WR5 este programat pe "1". Dacă se utilizează modul de transmisie asincron şi bitul RTS din WR5 este programat pe "0", ieşirea va trece pe "1" la golirea *buffer*-ului de transmisie. În modurile sincrone, conexiunea $\overline{RTS}$ urmăreşte starea bitului RTS din registrul WR5. Cele două ieşiri se pot întrebuinţa şi ca ieşiri de comandă de uz general.

$\overline{CTSA}$, $\overline{CTSB}$, *Clear To Send*, gata de emisie. Intrări active pe "0". Dacă bitul D_5, auto-validare, din registrul WR3, se programează pe "1", trecerile acestor linii pe "0" validează circuitele de transmisie respective. Programarea pe "0" a bitului D_5 permite folosirea lor ca intrări de uz general, ce pot fi citite în RR0. Ambele conexiuni sunt de tip *trigger-Schmitt*, permiţând comandarea lor cu semnale având fronturi lente. SIO poate detecta tranziţiile pozitive sau negative ale semnalelor aplicate la intrările CTS şi apoi, dacă a fost programat corespunzător, să genereze întreruperi către UC.

$\overline{DTRA}$, $\overline{DTRB}$, *Data Terminal Ready*, terminal de date gata. Ieşiri active pe "0" care urmăresc starea programată a bitului DTR din registrul WR5. De asemenea, ele pot fi folosite ca ieşiri de comandă de uz general.

$\overline{DCDA}$, $\overline{DCDB}$, *Data Carrier Detect*, detector purtătoare. Intrări active pe "0" de tip *trigger-Schmitt*, care, dacă se lucrează în modul auto-validare, bitul D_5 din WR3 programat pe "1", servesc la validarea recepţiei. Pot fi utilizate şi ca intrări de uz general, ţinând seama şi de faptul că SIO detectează tranziţiile de nivel, generând, dacă a fost programat, întreruperi către UC.

2.4.3. FUNCŢIONAREA CIRCUITULUI. POSIBILITĂŢI DE LUCRU

SIO-Z80 poate asigura comanda a două canale *duplex* independente, în multe din procedurile actual întâlnite în comunicaţiile de date sincrone sau asincrone. Funcţionarea circuitului este determinată de conţinutul registrelor de comandă, ce trebuie programate înainte ca SIO să execute vreo funcţie. Registrele de stare se pot investiga în orice moment, dar numai unele comenzi şi moduri pot fi modificate pe durata unei funcţionări deja programate.

2.4.3.1. **Moduri de lucru cu unitatea centrală**

Pentru realizarea transferului de date, stări şi comenzi de la şi spre UC, SIO-Z80 foloseşte testarea software, *polling*, întreruperile sau tehnica transferului pe blocuri de octeţi. Transferul pe blocuri de octeţi se poate efectua prin intermediul unităţii centrale, cu ajutorul instrucţiunilor I/E de transfer pe bloc, de exemplu OTIR, sau prin intermediul unor circuite specializate de tip DMA.

Tehnica testării software, polling, impune verificarea permanentă prin program a unor parametri specifici, ce caracterizează transferul. SIO-Z80 are asociate fiecărui canal două registre de stare, RR0, RR1, şi un registru comun cu vectorul de întrerupere modificat, RR2, accesibil numai pe canalul B. Cele două registre de stare sunt actualizate de logica internă a circuitului în momente definite în raport cu funcţiile realizate. Unul din registrele de stare se foloseşte pentru a indica unităţii centrale momentele când SIO are pregătite date sau când are nevoie de date. De asemenea, acelaşi registru memorează şi condiţiile de eroare. Al doilea registru de stare, ce păstrează condiţiile de recepţie speciale, nu trebuie citit decât după ce s-a recepţionat un caracter. În modul *polling* toate întreruperile circuitului se invalidează.

Lucrul în întreruperi cu SIO-Z80 este facilitat de modul elaborat în care circuitul poate genera întreruperi, permiţând utilizarea lui eficientă în aplicaţii de timp real. SIO are un registru, WR2, în care UC înscrie vectorul de întrerupere şi un altul RR2, ce păstrează vectorul modificat. Ambele registre sunt accesibile numai pe canalul B. Dacă circuitul se programează corespunzător, schimbarea de stare conduce la modificarea unei zone de trei biţi din vectorul de întrerupere astfel încât, acesta poate "puncta" direct opt rutine de tratare diferite, eliminându-se timpul consumat cu operaţiile de analiză a stării circuitului.

Întreruperile generate de SIO-Z80 se împart în trei grupe, de recepţie, de transmisie şi externe/de stare, în această ordine de prioritate pe fiecare canal, canalul A fiind prioritar faţă de canalul B. Fiecare sursă de generare a întreruperii poate fi validată prin program. Dacă se validează *întreruperile de transmisie*, UC va fi întreruptă la golirea *buffer*-ului de transmisie, după ce,

bineînțeles, el a fost încărcat cu un caracter. *Întreruperile de recepție* pot fi de două feluri: *întreruperi-la-primul-caracter-recepționat* și *întreruperi-la-fiecare-caracter-recepționat*.

Primul fel de întreruperi de recepție se utilizează de obicei în cazul modului de transfer pe blocuri de octeți. Întreruperea-la-fiecare-caracter-recepționat permite modificarea vectorului în situația detectării unei erori de paritate. Ambele tipuri de întrerupere admit generarea *întreruperilor datorită condițiilor-speciale-de-recepție*[1], spre exemplu, întrerupere-de-sfârșit-de-cadru în procedura SDLC. O condiție de recepție specială poate conduce la generarea unei întreruperi, numai dacă în prealabil a fost selectat unul din cele două tipuri de întrerupere, la primul caracter recepționat sau la fiecare caracter recepționat. În modul întrerupere-la-primul-caracter-recepționat, condițiile speciale de recepție, cu excepția erorii de paritate, generează întreruperi și după activarea întreruperii la recepția primului caracter, de pildă întrerupere de depășire la recepție. *Întreruperile externe de stare* au funcția de a supraveghea tranzițiile semnalelor de la intrările $\overline{CTS}$, $\overline{DCD}$ și $\overline{SYNC}$. O întrerupere externă/de stare poate fi cauzată de necesitatea de emisie a CRC sau de detecția, în șirul de date recepționate, a unei secvențe BREAK, în comunicațiile asincrone, ori a unei secvențe ABORT, în protocolul SDLC. Întreruperea declanșată de o secvență BREAK/ABORT permite circuitului SIO să atenționeze unitatea centrală la detecția sau la terminarea unei asemenea secvențe. Astfel, întreruperile externe/de stare înlesnesc terminarea corectă a mesajului curent, inițializarea următorului mesaj precum și detectarea oportună a secvențelor BREAK/ABORT.

Dacă unitatea centrală a sistemului e realizată cu microprocesorul Z80, lucrul în întreruperi se desfășoară conform modului 2, SIO generând vectorul în ciclul de achitare, vezi § 2.1, pentru a forma, împreună cu conținutul registrului I, adresa unei locații unde se află adresa de început a subrutinei de serviciu.

Transferul pe blocuri de octeți poate fi implementat fie cu ajutorul unității centrale, prin utilizarea instrucțiunilor de I/E pe bloc, de exemplu `OTIR`, `OTDR`, `INIR` sau `INDR` pentru Z80, fie prin folosirea unor circuite specializate pentru acces direct la memorie, DMA, ca DMA-Z80 sau 8257. Acest mod de lucru întrebuințează ieșirea $\overline{W/RDY}$, comandată cu ajutorul a trei biți programabili, D_7, D_6, D_5, din registrul de comandă WR1. Ieșirea poate îndeplini funcția $\overline{WAIT}$ pentru ca, prin conectarea la intrarea corespunzătoare a microprocesorului, să permită transferul pe bloc cu ajutorul instrucțiunilor specializate de I/E, prelungind ciclii de I/E atunci când SIO nu e pregătită de transfer. Când se utilizează circuite DMA, ieșirea realizează funcția READY indicând circuitului specializat că SIO este gata pentru un transfer cu memoria.

---

[1] Vezi și § 2.4.4.1

2.4.3.2. Moduri de transmisie asincrone

Transmisia şi recepţia se realizează independent pe fiecare din cele două canale. Caracterele pot avea 5÷8 biţi şi, opţional, un bit de paritate. Partea de transmisie a circuitului Z80 poate adăuga fiecărui caracter 1, 11/2 sau 2 biţi de STOP şi poate genera în orice moment caracterul BREAK. SIO nu impune folosirea unor semnale bază de timp pentru transmisie şi recepţie cu factor de umplere 1/2 ceea ce permite utilizarea unor circuite cum este şi CTC-Z80. Rata de transmisie a datelor se poate selecta pentru a fi 1/1, 1/16, 1/32 sau 1/64 din frecvenţa ceasului. Conexiunea externă $\overline{SYNC}$ este posibil de a fi programată ca intrare la care să se aplice un semnal de tipul *Detector de apel* generat de modem.

Transmisia începe numai după programarea pe "1" a bitului validare-transmisie, D_3, din registrul de comandă WR5[1]. Dacă a fost programat modul auto-validare, bitul D_5 din WR3, începerea transmisiei va fi condiţionată şi de activarea intrării $\overline{CTS}$. Starea de repaus, în care SIO nu emite, este starea MARK, "1", până când se iniţiază o transmisie. La programarea în WR5 a bitului D_4, transmisie-BREAK, ieşirea TxD este forţată în starea SPACE, "0", indiferent de starea în care se găsea înaintea acestei comenzi. Pentru a elibera linia, fie se anulează comanda transmisie-BREAK, fie se iniţializează circuitul. Mai precizăm că, dacă se utilizează un cod cu 5 biţi/caracter, biţii nefolosiţi ai fiecărui octet scris în SIO-Z80, D_5, D_6 şi D_7, vor fi "0".

Recepţia asincronă începe numai după ce bitul validare-recepţie, D_0, din registrul de comandă WR3, a fost programat pe "1". Dacă se foloseşte modul auto-validare, este necesar, pentru a valida recepţia, ca intrarea $\overline{DCD}$ să fie şi ea activată. Recepţia este protejată la tranziţiile pe "0" parazite, de scurtă durată, cu ajutorul unui mecanism de rejecţie care testează semnalul după o jumătate de bit de la detecţia nivelului "0" pe liniile de recepţie, RxDA sau RxDB. Dacă nivelul "0" nu se menţine, tranziţia este parazită şi procesul de asamblare a caracterului nu va mai fi declanşat. Această metodă de detecţie a unui bit de START micşorează numărul erorilor în recepţiile pe linii zgomotoase.

Circuitele de recepţie au, după cum se poate vedea şi în figura 2.44, o stivă cu patru niveluri, incluzând şi registrul de deplasare, pentru stocarea datelor primite. Această stocare permite tratarea de către UC a unei întreruperi de început de bloc, în cazul unui transfer de viteză mare. Există, de asemenea, o stivă pentru erori, paralelă cu stiva de date, indicatorii de eroare fiind încărcaţi o dată cu caracterele la care se referă. Indicatorii pot fi citiţi prin program aflându-se în registrul cu condiţiile speciale de recepţie, RR1[2]. Indicatorii eroare-depăşire-recepţie şi eroare-paritate sunt cumulativi şi nu pot

---

[1] Vezi § 2.4.4.1

[2] Vezi § 2.4.4.2.

fi şterşi decât prin comanda 6, iniţializare-erori[1]. Pe de altă parte, indicatorii sfârşit-de-cadru şi eroare-de-CRC/cadrare reflectă starea curentă a caracterului aflat în *buffer* şi nu sunt şterşi de o comandă iniţializare-erori. De aceea citirea stării indicatorilor de eroare, registrul RR1, va reflecta întotdeauna atât starea caracterului din *buffer*-ul de recepţie cât şi orice eroare de paritate sau depăşire apărută de la ultima comandă iniţializare-erori. Totodată, pentru a păstra corespondenţa între caracterele recepţionate şi indicatorii de eroare asociaţi, registrul de stare trebuie citit *înaintea* datelor, exceptând situaţiile ce vor fi descrise mai jos. Prin utilizarea întreruperilor vectorizate cerinţa se îndeplineşte uşor, deoarece SIO generează un vector de întrerupere special, în cazul apariţiei unei erori sau a sfârşitului de cadru.

Este posibil ca în starea citită după date să fie incluse şi erorile următorului caracter, în cazul când acesta a fost recepţionat. În situaţia în care operaţiile de citire se execută suficient de rapid, suprapunerea indicatorilor de stare nu mai apare. O excepţie intervine dacă se lucrează în modul întrerupere-la-primul-caracter-recepţionat deoarece, în acest context, o întrerupere generată de o condiţie specială face ca erorile şi caracterul, chiar dacă au fost deja citite, să fie păstrate până la o comandă iniţializare-erori. Astfel se previne recepţia unor caractere noi până la trimiterea unei comenzi de iniţializare. De asemenea, selectând modul întrerupere-la-primul-caracter vectorul de întrerupere emis de SIO într-un ciclu de achitare va fi modificat de apariţia unei erori. La depăşire, ultimul caracter se încarcă peste caracterul recepţionat anterior, care în acest fel se pierde. În acelaşi timp se poziţionează indicatorul eroare-depăşire-recepţie şi se modifică vectorul pe starea condiţie-de-recepţie-specială, dacă a fost în prealabil validat modul de lucru starea-afectează-vectorul.

Când se lucrează în modul *polling*, pentru ca UC să preia un caracter recepţionat, va fi necesară testarea bitului caracter-recepţionat-disponibil, D_0, din registrul RR0. Acest bit este pus pe "0" de logica internă a lui SIO când *buffer*-ele de recepţie sunt goale. La transmisie interesează bitul *buffer*-transmisie-gol, D_2 din RR0, pus pe "1" de SIO ori de câte ori *buffer*-ul de transmisie nu are nici un caracter. Înaintea fiecărei scrieri în acest *buffer* UC trebuie să testeze bitul D_2 din RR0, pentru a se asigura că nu apare o suprapunere de caractere.

2.4.3.3. **Moduri de transmisie sincrone**

SIO-Z80 permite implementarea atât a protocoalelor orientate pe caracter, BCP, *Byte Control Protocol*, cât şi a celor orientate pe bit, BOP, *Bit Oriented Protocol*, [27]. Protocoalele BCP sunt de tip MONO-SYNC, cu un singur caracter de sincronizare de 8 biţi, BISYNC, cu secvenţă de sincronizare de 16

---

[1] Vezi § 2.4.4.1.

biţi sau pot fi cu sincronizare externă. Caracterele de sincronizare se detectează fără a fi necesară întreruperea UC. SIO poate detecta şi caractere de sincronizare de 5, 6 sau 7 biţi. La BCP verificarea CRC se întârzie cu un caracter, astfel încât UC poate invalida calculul CRC pentru anumite caractere, ceea ce permite implementarea unor protocoale ca BISYNC al firmei IBM. Circuitu poate calcula resturi utilizând două polinoame generatoare: CRC-16, $x^{16} + x + x^2 + 1$, şi CCITT, $x^{16} + x^{12} + x^5 + 1$. În modul SDLC, iniţializarea se face cu "1", iar în celelalte cu "0". Dacă nu mai sunt date de transmis, SIO poate transmite automat caracterele CRC, astfel creîndu-se posibilitatea folosirii circuitului în transmisii de viteză mare prin DMA, fără intervenţia UC la sfârşitul unui mesaj. Când nu sunt nici date nici caractere CRC disponibile, SIO inserează caractare SYNC de 8 sau 16 biţi, indiferent de lungimea programată a caracterelor.

Protocoalele BOP, cum sunt SDLC sau HDLC, se implementează comod cu SIO-Z80, circuitul asigurând transmiterea automată a delimitatorului, inserarea şi eliminarea biţilor "0", generarea CRC. Printr-o comandă specială se poate emite secvenţa ABORT. La sfârşitul mesajului, când *buffer*-ul de transmisie devine gol, circuitul transmite automat caracterele CRC şi delimitatorul. Dacă apare o eroare de ritm în timpul transmisiei mesajului, o întrerupere externă de stare poate avertiza UC, aceasta având posibilitatea să comande generarea unei secvenţe ABORT. Se pot transmite caractere de $1 \div 8$ biţi şi se pot recepţiona mesaje, fără a se fi precizat în prealabil structura caracterelor în interiorul câmpului de informaţie al unui cadru.

La recepţia de tip BOP circuitul se sincronizează automat pe delimitatorul de început al unui cadru, generând un semnal de sincronizare la conexiunea externă $\overline{SYNC}$ sau, dacă a fost programat, o întrerupere. SIO mai poate fi programat cu ajutorul câmpului de adresă, pentru a căuta cadre adresate unui utilizator specificat, sau cadre emise către toţi utilizatorii. Cadrele care nu îndeplinesc nici una din cele două condiţii vor fi ignorate. Numărul de caractere din câmpul de adresă poate fi extins prin program. Dacă se lucrează în întreruperi, se pot selecta cele două moduri amintite mai sus: întrerupere-la-primul-caracter-recepţionat sau întrerupere-la-fiecare-caracter-recepţionat.

Toate modurile sincrone impun folosirea unei rate de transfer a datelor egale cu frecvenţa ceasului, atât la transmisie cât şi la recepţie. Datele vor fi eşantionate la recepţie cu frontul pozitiv al ceasului $\overline{TxC}$, în timp ce la transmisie schimbările de biţi vor apărea pe frontul negativ al ceasului $\overline{TxC}$. După o iniţializare hardware sau software circuitele de recepţie se vor afla în modul aşteptare-sincronizare, *hunt*, activat de fapt după o comandă validare-recepţie. Transferul datelor va începe numai după realizarea sincronizării. La pierderea sincronizării se poate reintra în modul aşteptare-sincronizare prin programarea pe "1" a bitului D_4 din registrul de comandă WR3. Diferenţele între protocoalele de tip MONOSYNC, BISYNC şi cu sincronizare externă sunt date numai de maniera în care se face sincronizarea; modul de funcţionare trebuie să fie selectat înaintea încărcării caracterelor de sincronizare, deoarece registrele sunt folosite diferenţiat:

– în modul MONOSYNC, caracter SYNC de 8 biţi, validarea transferului de date se va face după detecţia unui singur caracter SYNC programat în registrul WR7;

– în modul BISYNC, sincronizare pe 16 biţi, asamblarea caracterelor va începe după detecţia a două caractere de sincronizare adiacente programate în registrele WR6 şi WR7;

– în modul cu sincronizare externă, asamblarea caracterelor va începe cu primul front pozitiv al ceasului $\overline{\text{RxC}}$ după activarea conexiunii externe $\overline{\text{SYNC}}$ care va trebui menţinută pe "0" cel puţin trei perioade de ceas.

În modurile MONOSYNC şi BISYNC conexiunea externă $\overline{\text{SYNC}}$ e folosită ca ieşire şi se va activa ori de câte ori va fi detectată o secvenţă de sincronizare, fiind ţinută "0" pe durata perioadei de ceas în care s-a făcut detecţia. În toate cele trei moduri asamblarea caracterelor va continua până la iniţializarea circuitului SIO, până la invalidarea recepţiei, printr-o comandă sau cu ajutorul semnalului $\overline{\text{DCD}}$, dacă se lucrează cu auto-validare, sau până când UC programează modul aşteptare-sincronizare.

2.4.3.3.1. *Transmisia sincronă de tip BCP*

După o iniaţializare sau când transmiterea nu este încă validată, circuitul SIO-Z80 menţine linia de emisie în starea MARK. La programarea bitului transmisie-BREAK, D_4 din WR5, această linie va fi trecută în starea SPACE, indiferent de conţinutul registrului de transmisie. În urma validării transmisiei şi a selecţiei modului, în starea de repaus, circuitul va transmite continuu pe linie caracterul de sincronizare programat, de 8 sau 16 biţi.

În transmisie, SIO poate fi utilizat în întreruperi sau în modul *polling*. Dacă se folosesc, întreruperile pot fi de transmisie sau externe/de stare. Programând bitul validare-întreruperi-transmisie, D_1 din WR1, SIO va genera o întrerupere de fiecare dată când *buffer*-ul de transmisie se goleşte. Această întrerupere poate fi achitată fie prin scrierea unui nou caracter în *buffer*, fie cu ajutorul comenzii iniţializare-întrerupere-transmisie-în-aşteptare, comanda 5. Când întreruperea se achită cu această comandă şi dacă nu se mai scrie nici un caracter în *buffer*, SIO nu va mai genera o altă întrerupere de transmisie. După scrierea unui caracter, *buffer*-ul se poate goli din nou, generându-se astfel o întrerupere de transmisie. Dacă se programează bitul validare-întreruperi-externe/de stare, D_0 din WR1, întreruperea se activează în situaţiile: începerea transmisiei caracterelor CRC sau a caracterelor SYNC, schimbări de stare la intrările $\overline{\text{DCD}}$, $\overline{\text{SYNC}}$ şi $\overline{\text{CTS}}$. Aceste tipuri de întrerupere pot avea asociate un vector unic, în situaţia în care a fost în prealabil selectat modul de lucru starea-afectează-vectorul, D_2 din WR1.

Atunci când nu mai sunt date de transmis, SIO va insera automat caractere SYNC, dacă transmisia CRC nu a fost validată. Generarea unei întreruperi se va face numai după ce s-a încărcat primul dintre caracterele SYNC inserate. Dacă transmisia CRC este validată în momentul în care nu mai sunt date de

emis, SIO va transmite caracterele CRC, 16 biţi, urmate de caractere SYNC. Pe timpul cât se transmite restul CRC, indicatorul eroare-de-ritm-la-transmisie/EOM, D_6 în RR0, se pune pe "1"; bitul *buffer*-transmisie-gol rămânând pe "0", pentru a semnifica starea *buffer*-ului încă plin. CRC nu se calculează pentru caracterele SYNC inserate automat, dar va fi calculat pentru orice caracter SYNC emis ca date, exceptând situaţia în care generatorul CRC este invalidat la încărcarea caracterului din *buffer* în registrul de deplasare pentru transmisie. După transmiterea caracterelor CRC, indicatorul *buffer*-transmisie-gol trece pe "1", generându-se şi o întrerupere ce semnalează unităţii centrale posibilitatea de a începe un nou mesaj. Comanda generatorului CRC se poate face ţinând cont că, înainte de încărcarea datelor, va trebui iniţializat prin trimiterea unei comenzi iniţializare-generator-CRC-la-transmisie; datele pot fi încărcate numai după validarea generării CRC şi a transmisiei. Înainte de a se emite caracterele CRC, dar, cel mai devreme după încărcarea primului caracter, trebuie trimisă comanda iniţializare-eroare-de-ritm-la-transmisie/EOM. Cât timp eroare-de-ritm-la-transmisie/ EOM este pe "1" transmisia CRC este inhibată, circuitul SIO putând să insereze în cadrul mesajelor, când nu mai sunt date disponibile, caracterul SYNC. La sfârşitul mesajului însă, acest bit trebuie pus pe "0", pentru a se permite emisia CRC.

Invalidarea transmisiei pe timpul emisiei unui caracter conduce, după transmiterea normală a caracterului, la trecerea liniei de emisie în starea MARK. De asemenea, dacă există un caracter în *buffer*, caracterul se păstrează. Pe de altă parte, la trimiterea de către UC a unei comenzi transmisie-BREAK, secvenţa BREAK, trecerea liniei în starea SPACE, va fi imediat emisă, caracterele în curs de transmisie pierzându-se.

2.4.3.3.2. *Recepţia sincronă de tip BCP*

Validarea recepţiei se face după programarea modului şi transmiterea caracterelor SYNC, în această ordine. În continuare, circuitele de recepţie se vor găsi în starea aşteptare-sincronizare în care vor rămâne până la: detecţia unui caracter SYNC, în modul MONOSYNC, detecţia a două caractere SYNC, în modul BISYNC, sau activarea pe "0" a conexiunii externe $\overline{SYNC}$. În primele două cazuri această conexiune funcţionează ca ieşire pentru a indica sincronizarea, iar în ultimul caz, ca intrare.

Asamblarea caracterelor începe imediat, după detecţia secvenţei de sincronizare. La recepţie, ca şi la emisie, SIO poate lucra cu sau fără întreruperi. Dacă se lucrează în întreruperi, acestea pot fi întrerupere-la-primul-caracter-recepţionat sau întrerupere-la-fiecare-caracter-recepţionat. Primul tip de întrerupere se utilizează, de obicei, pentru a lansa fie o buclă de testare software, fie o instrucţiune I/E de transfer pe bloc, întrebuinţând în ultimul caz ieşirea $\overline{W/RDY}$ pentru sincronizarea UC cu SIO. De asemenea, o astfel de întrerupere poate fi folosită şi pentru a lansa un circuit specializat de acces direct la memorie, DMA. Lucrând cu acest tip de întrerupere, SIO-Z80 va

întrerupe la primul caracter recepţionat şi, apoi, numai în caz de eroare. Anularea acestui mod de întrerupere se face cu ajutorul comenzii 4, iniţializare-întrerupere-la-primul-caracter-recepţionat. Al doilea tip de întrerupere face ca SIO să activeze linia $\overline{INT}$ ori de câte ori în *buffer*-ul de recepţie se găseşte un caracter asamblat. Condiţiile speciale de recepţie sau erorile pot genera un vector unic, dacă în prealabil a fost selectat modul starea-afectează-vectorul. Eroarea de paritate poate, opţional, să nu conducă la generarea unui vector special.

Calculul CRC pentru un anumit caracter începe cu o întârziere de 8 cicli de ceas după ce caracterul a fost transferat din registrul de deplasare în stiva de recepţie (a se vedea şi figura 2.44). Validarea/invalidarea CRC înaintea transferării unui caracter în stivă face ca precedentul caracter transferat în stivă să fie inclus/exclus din canalul CRC. Astfel, verificarea CRC poate fi comandată selectiv, existând posibilitatea ca anumite caractere să nu intre în calculul CRC.

2.4.3.3.3. *Transmisia sincronă de tip BOP (SDLC)*

Starea de repaus a liniei de emisie, înainte de a valida transmisia, este MARK. După validarea transmisiei, pe această linie, în starea de repaos, se vor genera în mod continuu delimitatoare. Pentru a transmite un bloc de date este necesară mai întâi iniţializarea generatorului CRC cu ajutorul comenzii iniţializare-generare-CRC-la- transmisie. Această iniţializare poate fi făcută oricând după emisia caracterelor CRC corespunzătoare mesajului precedent. Ca şi în cazul BCP, pe durata transmisiei restului CRC indicatorul eroare-de-ritm-la-transmisie/EOM se pune pe "1" iar *buffer*-transmisie-gol rămâne pe "0". La sfârşitul transmisiei CRC, al doilea indicator se poziţionează şi el pe "1", conducând la generarea unei întreruperi. Secvenţa CRC se transmite automat atunci când transmiţătorul nu mai are date de emis. După transmisia ei pe linie se vor emite delimitatoare. Generarea unei secvenţe ABORT se face prin comanda 1, transmisie-ABORT. Aceasta va conduce la generarea a $8 \div 14$ biţi "1", după care se va trece în starea de repaus emiţându-se în mod continuu delimitatoare. Caracterele în curs de transmisie sau în aşteptare în *buffer* se vor pierde. Inserarea de biţi "0" se face automat de către SIO după întâlnirea a cinci biţi "1" succesivi, exceptând secvenţele delimitator şi ABORT.

Se pot transmite caractere de $1 \div 8$ biţi, numărul de biţi/caracter putând fi schimbat pe timpul transmisiei unui bloc. În felul acesta se asigură că lungimea câmpului de date, I, dintr-un cadru, să fie variabilă la nivel de bit. Utilizând codul-de-rest-I-la-recepţie, SIO are posibilitatea să recepţioneze un mesaj având orice lungime în biţi şi să-l retransmită exact cum l-a primit fără să fie necesare, în prealabil, alte informaţii despre structura caracterelor din câmpul I. Schimbarea numărului de biţi/caracter nu afectează caracterul deplasat

în registrul de transmisie, fiecare caracter fiind serializat cu numărul de biţi programat în momentul transferării din *buffer* în registrul de transmisie.

2.4.3.3.4. *Recepţia sincronă de tip BOP (SDLC)*

Transferul de date între SIO şi UC poate începe după primirea primului caracter diferit de un delimitator şi după recepţia cel puţin a unui delimitator, cu condiţia să nu se lucreze în modul căutare-adresă, D_2 din WR3. Dacă se lucrează în acest mod pentru lansarea transferului, mai este necesar ca, după delimitator, să urmeze fie adresa programată, fie adresa globală. Când nu se foloseşte modul căutare-adresă sau în cazul în care câmpul de adresă este format din mai mulţi octeţi, există posibilitatea ca un mesaj nedorit să nu fie citit complet de UC. După luarea unei astfel de decizii de către UC, prin programarea bitului D_4 din WR3, intrare-în-modul-aşteptare-sincronizare, SIO va suspenda recepţia până la următorul delimitator de început al unui cadru.

Desfăşurarea transferului poate fi comandată prin *polling* sau prin întreruperi. În prima situaţie, disponibilitatea unui caracter în *buffer*-ul de recepţie se detectează cu ajutorul indicatorului caracter-recepţionat-disponibil, D_0 din RR0. În a doua situaţie, prin selectarea întreruperii-la-primul-caracter-recepţionat, se asigură o cale de începere a unui transfer de bloc. Pentru ieşirea din bucla de transfer, dacă nu se cunoaşte lungimea cadrului, se poate folosi o întrerupere pe o condiţie-de-recepţie-specială, sfârşit-de-cadru, generată la recepţia delimitatorului de terminare a unui cadru. Pentru situaţiile în care numărul de biţi din câmpul de date nu este un multiplu întreg al lungimii caracterelor, lungime programată cu biţii D_7, D_6 din WR3, utilizatorul va trebui să ţină cont şi de valoarea codului-de-rest-I-la-recepţie. Rearmarea întreruperii la primul caracter al următorului cadru se face prin transmiterea comenzii 4, iniţializare-întrerupere-la-primul-caracter.

Biţii "0", inseraţi la transmisie pentru a elimina din câmpul de date secvenţele-delimitator, sunt rejectaţi automat la recepţie. Detectarea unei secvenţe ABORT, mai mult de 7 biţi "1", conduce la punerea pe "1" a indicatorului BREAK/ABORT din RR0 şi generarea, dacă a fost validată, a unei întreruperi externe/de-stare. Reactivarea acestei întreruperi se va face cu comanda 2, iniţializare-întrerupere-externă/de-stare. Lungimea caracterelor poate fi modificată de UC chiar pe timpul recepţiei unui mesaj.

Validarea verificării CRC se face automat, calculul fiind iniţializat cu delimitatorul de început al unui cadru şi continuat până la întâlnirea delimitatorului final. Rezultatul verificării CRC este dat de indicatorul eroare-de-CRC/cadrare din RR1, "0" pentru mesaj corect recepţionat. Verificarea de paritate poate fi utilizată pentru datele din câmpul I, dacă acestea sunt reprezentate prin caractere de $5 \div 7$ biţi şi se utilizează o legătură semi-duplex, neexistând circuite separate pentru controlul parităţii la emisie şi recepţie.

2.4.4. PROGRAMAREA CIRCUITULUI SIO-Z80

Programarea circuitului SIO-Z80 impune transmiterea unor comenzi pentru selectarea modului de comunicaţie şi, apoi, pentru stabilirea parametrilor specifici de lucru cu UC şi modemul. Personalizarea funcţionării circuitului înseamnă, în principal, scrierea registrelor de comandă.

2.4.4.1. **Registrele de comandă**

Pentru a fi programat, oricare din cele şapte sau opt registre de comandă ale canalului A, respectiv B, necesită scrierea de către UC a doi octeţi. Excepţie face numai registrul WR0. O iniţializare hardware sau software va poziţiona logica de adresare a registrelor pe WR0 astfel încât primul octet de comandă transmis va fi înscris în acest registru. De aceea, WR0 conţine principalele comenzi şi câmpul de iniţializare CRC. De asemenea, biţii D_2, D_1 şi D_0 din WR0 specifică adresa registrului în care se va înscrie octetul de comandă următor.

Registrul de comandă WR0 are următorul format:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Cod iniţializare CRC		Comandă			Adresă registru următor		

Adresa registrului următor este numărul registrului exprimat în binar cu ajutorul biţilor D_2, D_1, D_0.

Cele opt comenzi care se pot transmite unui canal SIO sunt codificate conform tabelului:

Comanda	D_5	D_4	D_3	Denumirea comenzii
0	0	0	0	Comandă-neoperantă
1	0	0	1	Transmisie-ABORT (modul SDLC)
2	0	1	0	Iniţializare-întreruperi-externe/de-stare
3	0	1	1	Iniţializare-canal
4	1	0	0	Iniţializare-întrerupere-la-primul-caracter-recepţionat
5	1	0	1	Iniţializare-întrerupere-transmisie-în-aşteptare
6	1	1	0	Iniţializare-erori
7	1	1	1	Revenire-din-întrerupere (numai canalul A)

Comanda 0, comandă-neoperantă, nu are nici un efect, fiind folosită la transmiterea adresei registrului următor.

Comanda 1, transmisie-ABORT, utilizată numai în procedura SDLC, pentru a iniţia generarea unei secvenţe de 8÷14 biţi "1".

Comanda 2, iniţializare-întreruperi-externe/de-stare, validează o nouă poziţionare a biţilor de stare din RR0, după generarea unei întreruperi externe/de stare, care blocase aceşti biţi pentru a-i menţine stabili până la citirea lor de către UC.

Comanda 3, iniţializare-canal, realizează aceeaşi iniţializare ca în cazul activării intrării $\overline{RESET}$, dar numai pe un canal. Trimisă pe canalul A, iniţializează şi logica de priorităţi a întreruperilor. După transmiterea acestei comenzi, până la următoarea comandă, este necesară trecerea a cel puţin 4 cicli de ceas.

Comanda 4, iniţializare-întrerupere-la-primul-caracter-recepţionat, reactivează acest tip de întrerupere, dacă a fost selectat, pregătind recepţia următorului mesaj.

Comanda 5, iniţializare-întrerupere-transmisie-în-aşteptare, previne generarea întreruperilor de transmisie, condiţionate de golirea *buffer*-ului, după terminarea transmiterii datelor, până la scrierea de către UC a unui nou caracter în *buffer*-ul de transmisie.

Comanda 6, iniţializare-erori, şterge eventualele erori de paritate şi/sau depăşire memorate în RR1 de la precedenta iniţializare.

Comanda 7, revenire-din-întrerupere, trimisă numai pentru canalul A, este interpretată de SIO ca o instrucţiune RETI conducând la ştergerea bistabilului "Întrerupere în curs de tratare", ceea ce permite validarea întreruperilor de la dispozitivele cu prioritate scăzută. Existenţa acestei comenzi asigură folosirea priorităţilor interne din SIO şi în sistemele care nu se bazează pe familia Z80.

Codul iniţializare CRC, D_7, D_6, are semnificaţia din tabelul de mai jos:

D_7	D_6	Denumirea comenzii
0	0	Comandă neoperantă
0	1	Iniţializare-verificare-CRC-la-recepţie
1	0	Iniţializare-generare-CRC-la-transmisie
1	1	Iniţializare-eroare-de-ritm-la-transmisie/EOM

Registrul de comandă WR1 are următorul format:

D_7	D_6	D_5	D_4 D_3	D_2	D_1	D_0
Validare WAIT/ READY	WAIT/ READY	$\overline{W}/\overline{RDY}$ la recepție/ transmisie	Mod de întrerupere la recepție	Starea afectează vectorul (numai canal B)	Validare întreruperi transmisie	Validare întreruperi externe/ de stare

Validare-întreruperi-externe/de-stare activează generarea întreruperilor, ca urmare a unor tranziții la intrările $\overline{DCD}$, $\overline{CTS}$ și $\overline{SYNC}$, a întâlnirii unei secvențe BREAK sau la începutul transmiterii caracterelor CRC ori SYNC. Dacă intrările enumerate nu se folosesc, ele vor trebui conectate la +5V, pentru a se preveni apariția unor întreruperi false.

Validare-întreruperi-transmisie permite activarea întreruperii ori de câte ori *buffer*-ul de transmisie devine gol.

Starea-afectează-vectorul, bit programabil numai pentru canalul B, care, dacă este pus pe "1", permite selectarea facilității ca vectorul trimis de SIO într-un ciclu de achitare-întrerupere să fie modificat conform tabelului:

V_3	V_2	V_1	Semnificația modificării	
0	0	0	*Buffer*-transmisie-gol	Canal B
0	0	1	Schimbare-externă/de-stare	
0	1	0	Caracter-recepționat-disponibil	
0	1	1	Condiție-de-recepție-specială	
1	0	0	*Buffer*-transmisie-gol	Canal A
1	0	1	Schimbare-externă/de-stare	
1	1	0	Caracter-recepționat-disponibil	
1	1	1	Condiție-de-recepție-specială	

Dacă $D_2=0$ vectorul trimis de SIO este cel scris prin program în registrul WR2.

Selectarea *modului-de-întrerupere-la-recepție* se face cu ajutorul biților D_4 și D_3:

D_4	D_5	Mod de întrerupere la recepţie
0	0	Întrerupere-recepţie-invalidată
0	1	Întrerupere-la-primul-caracter-recepţionat
1	0	Întrerupere-la-fiecare-caracter-recepţionat/ Paritatea-afectează-vectorul
1	1	Întrerupere-la-fiecare-caracter-recepţionat/ Paritatea-nu-afectează-vectorul

$\overline{W}/\overline{RDY}$-la-recepţie/transmisie selectează, dacă ieşirea $\overline{W}/\overline{RDY}$ este validată, situaţia în care această conexiune externă va fi activată: când *buffer*-ul de recepţie este gol, $D_5=1$, sau când *buffer*-ul de transmisie este plin, $D_5=0$.

WAIT/READY se programează pe "0", dacă linia $\overline{W}/\overline{RDY}$ se utilizează pentru a comanda o intrare de tip WAIT a unui microprocesor, şi pe "1" dacă se conectează la o intrare de tip READY a unui circuit specializat pentru DMA. Funcţia READY poate fi realizată de SIO oricând, nefiind condiţionată de selectarea circuitului, în timp ce funcţia WAIT este activă numai când UC încearcă să citească date din SIO, în momentul când *buffer*-ul de recepţie este gol, sau să scrie când *buffer*-ul de transmisie este plin.

Cât timp bitul *validare*-WAIT/READY este menţinut pe "0" ieşirea $\overline{W}/\overline{RDY}$ va fi "1" pentru modul READY şi în starea a treia pentru modul WAIT. $D_7=1$ va însemna validarea funcţionării ieşirii $\overline{W}/\overline{RDY}$ într-unul din cele două moduri.

Registrul de comandă WR2 este registrul care memorează vectorul de întrerupere al circuitului şi nu există decât în setul de registre al canalului B. Într-un ciclu de achitare a întreruperii, SIO va returna un vector având biţii $V_7 \div V_4$ şi V_0 la fel cu cei programaţi şi dacă a fost programat bitul starea-afectează-vectorul, cu V_3, V_2, V_1 modificaţi ca în tabelul de mai sus[1]. Formatul acestui registru este:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0
Se returnează nemodificaţi				Modificaţi numai dacă D_2/WR1 = 1 (canal B)			Nemo-dificat

[1] Vezi structura registrului WR1.

Registrul de comandă WR3 conține comenzi și parametri referitori la logica de recepție:

D_7 D_6	D_5	D_4	D_3	D_2	D_1	D_0
Lungime caracter la recepție	Auto-validare	Așteptare sincronizare	Validare CRC la recepție	Căutare adresă	Invalidare încărcare caracter SYNC	Validare recepție

Validare-recepție pe "1" permite începerea operațiilor de recepție.

Programând *invalidare-încărcare-caracter-SYNC*, caracterele SYNC, ce preced un mesaj nu se vor mai transfera în *buffer*-ul de recepție.

Prin selecția modului *căutare-adresă*, dacă se lucrează în protocolul SDLC, SIO va rejecta toate cadrele care nu au adresa egală cu cea programată sau cu adresa de emisie globală (*broadcast*).

Punerea pe "1" a bitului *validare-CRC-la-recepție* conduce la o reluare sau inițiere a calculului CRC începând cu ultimul caracter transferat din registrul de recepție în stivă, fără a se mai ține cont de eventualele caractere precedente păstrate în stivă.

Programarea pe "1" a bitului *așteptare-sincronizare* permite reintrarea în modul de așteptare a secvenței de sincronizare în cazurile când s-a pierdut sincronizarea sau, într-un protocol de tip BOP, când UC consideră că mesajul primit nu este necesar.

Selectând modul *auto-validare*, $D_5=1$, intrările $\overline{DCD}$ și $\overline{CTS}$ vor fi folosite pentru validarea circuitelor de recepție, respectiv transmisie. $D_5=0$ inhibă această funcție menținând doar posibilitatea ca starea intrărilor să fie citită în registrul RR0.

Lungimea caracterelor asamblate la recepție este determinată de valoarea biților D_7, D_6, conform tabelului:

D_7	D_6	Număr de biți/caracter
0	0	5
0	1	7
1	0	6
1	1	8

Numărul de biți/caracter poate fi reprogramat în timpul asamblării.

Registrul de comandă WR4, conţinând comenzi ce afectează atât transmisia cât şi recepţia, are următorul format:

D_7 D_6	D_5 D_4	D_3 D_2	D_1	D_0
Rata datelor	Moduri de sincronizare	Număr de biţi STOP	Paritate pară/ impară	Paritate

Dacă *paritate* este pus pe "1" SIO adaugă la numărul de biţi specificaţi în D_7, D_6/WR3, bitul de paritate. De asemenea, dacă $D_0=1$, la recepţie se ia în considerare şi bitul de paritate.

Paritate-pară/impară specifică atât pentru transmisie cât şi pentru recepţie felul parităţii: $D_1=0$, paritate impară, $D_1=1$ paritate pară.

Numărul-de-biţi-STOP adăugaţi la transmisia asincronă a caracterelor se fixează cu ajutorul biţilor D_3 şi D_2:

D_3	D_2	Număr de biţi STOP
0	0	Moduri sincrone
0	1	1 bit STOP pe caracter
1	0	1 1/2 biţi STOP pe caracter
1	1	2 biţi STOP pe caracter

Selecţia *modului-de-sincronizare* se face cu D_5, D_4:

D_5	D_4	Moduri de sincronizare
0	0	Caracter SYNC de 8 biţi transmis prin program
0	1	Caracter SYNC de 16 biţi transmis prin program
1	0	Mod SDLC (secvenţa de sincronizare 01111110)
1	1	Sincronizare externă

Biţii D_7, D_6 se utilizează pentru specificarea ratei de transmisie a datelor, în funcţie de frecvenţa ceasului de emisie/recepţie. Pentru modurile sincrone

rata datelor trebuie să fie aleasă egală cu frecvenţa ceasului, în timp ce pentru modurile asincrone se poate selecta orice raport. Raportul ales va fi folosit de SIO atât pentru emisie cât şi pentru recepţie. Precizăm că pentru oricare mod de lucru se va avea în vedere ca frecvenţa ceasului de sistem, Φ, să fie de cel puţin 5 ori mai mare decât rata datelor. De asemenea, pentru o rată a datelor egală cu frecvenţa ceasului sincronizarea de bit trebuie asigurată extern. Selecţia *ratei datelor* se face conform tabelului:

D_7	D_6	Rata datelor
0	0	Rata datelor x 1 = Frecvenţa ceasului
0	1	Rata datelor x 16 = Frecvenţa ceasului
1	0	Rata datelor x 32 = Frecvenţa ceasului
1	1	Rata datelor x 64 = Frecvenţa ceasului

Registrul de comandă WR5 conţine cea mai mare parte a comenzilor şi parametrilor ce afectează transmisia:

D_7	D_6 D_5	D_4	D_3	D_2	D_1	D_0
DTR	Lungime caracter la transmisie	Transmisie BREAK	Validare transmisie	CCITT/ CRC-16	RTS	Validare CRC la transmisie

Validare-CRC-la-transmisie permite, dacă este pus pe "1", transmisia automată a caracterelor CRC atunci când nu mai sunt date de emis. Cu ajutorul acestui bit de comandă se poate valida/invalida calculul CRC pentru orice caracter.

RTS comandă ieşirea $\overline{RTS}$: $D_1=1$ conduce la activarea, trecerea pe "0", a ieşirii $\overline{RTS}$ în timp ce $D_1=0$ permite trecerea pe "1" a ieşirii $\overline{RTS}$ dar numai după ce transmiţătorul este gol.

CCITT/CRC-16 selectează polinomul generator utilizat în verificarea CRC, la emisie şi recepţie: "0" pentru polinomul CCITT, $x^{16} + x^{12} + x^5 + 1$.

Cât timp bitul *validare-transmisie* este "0" ieşirea TxD va fi ţinută în starea MARK, "1". Emisia va începe după trecerea bitului pe "1". Punerea pe "0" a acestei comenzi va încheia transmisia după terminarea caracterului curent sau imediat, dacă invalidarea se face pe timpul emisiei unui caracter CRC.

Poziţionarea pe "1" a comenzii *transmisie-BREAK* forţează linia TxD în starea SPACE, "0".

Numărul de biţi dintr-un octet transferat în *buffer*-ul de transmisie, ce vor fi deci emişi în linie, este selectat cu ajutorul biţilor D_6 şi D_5:

D_6	D_5	Număr de biţi/caracter
0	0	5 sau mai puţini
0	1	7
1	0	6
1	1	8

Biţii care se emit se presupun aliniaţi la dreapta octetului, transmisia începând cu bitul cel mai puţin semnificativ, D_0. În cazul transmiterii a 5 sau mai puţini biţi structura octetului va fi:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Număr biţi/ caracter
1	1	1	1	0	0	0	D	1 bit
1	1	1	0	0	0	D	D	2 biţi
1	1	0	0	0	D	D	D	3 biţi
1	0	0	0	D	D	D	D	4 biţi
0	0	0	D	D	D	D	D	5 biţi

DTR este bitul de comandă al ieşirii $\overline{DTR}$: $D_7=1$ conduce la activarea, punerea pe "0", a ieşirii $\overline{DTR}$, $D_7=0$ face ca $\overline{DTR}=1$.

Registrul de comandă WR6 poate conţine primii 8 biţi ai unui caracter de sincronizare de 16 biţi (BISYNC), o adresă, dacă se utilizează modul căutare-adresă, sau caracterul SYNC dacă a fost selectat modul de sincronizare pe caracter SYNC de 8 biţi:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
$SYNC_7$	$SYNC_6$	$SYNC_5$	$SYNC_4$	$SYNC_3$	$SYNC_2$	$SYNC_1$	$SYNC_0$
AD_7	AD_6	AD_5	AD_4	AD_3	AD_2	AD_1	AD_0

Registrul de comandă WR7 poate conţine al doilea octet al caracterului de sincronizare de 16 biţi, caracterul de sincronizare de 8 biţi sau, pentru protocolul SDLC, secvenţa 01111110. Cele două registre, WR6 şi WR7, nu se utilizează în modul de sincronizare externă. Formatul lui WR7 este:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
$SYNC_{15}$	$SYNC_{14}$	$SYNC_{13}$	$SYNC_{12}$	$SYNC_{11}$	$SYNC_{10}$	$SYNC_9$	$SYNC_8$
0	1	1	1	1	1	1	0

2.4.4.2. **Registrele de stare**

SIO-Z80 are, pentru memorarea stării fiecărui canal, câte trei registre de stare: RR0, RR1, şi RR2. RR2, vectorul de întrerupere modificat, nu poate fi citit decât pe canalul B. Citirea unui registru de stare presupune două operaţii de I/E: întâi scrierea adresei registrului în WR0 şi apoi citirea propriu-zisă.

Registrul de stare RR0 are următorul format:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
BREAK/ ABORT	Eroare de ritm la transmisie/ EOM	CTS	SYNC/ aşteptare sincroni-zare	DCD	*Buffer* transmisie gol	Întrerupere în aşteptare (numai canalul A)	Caracter recepţionat disponibil

Caracter-recepţionat-disponibil este pus pe "1" când în *buffer*-ul de recepţie se află cel puţin un caracter.

Indicatorul *întrerupere-în-aşteptare*, accesibil numai pe canalul A, fiind întotdeauna "0" pe canalul B, reflectă starea întreruperilor pentru întregul circuit.

Buffer-transmisie-gol se pune pe "1" ori de câte ori *buffer*-ul de transmisie devine gol, cu excepţia situaţiei în care se transmit caracterele CRC în modurile sincrone.

DCD indică starea intrării $\overline{DCD}$ în timpul ultimei schimbări a oricărui indicator extern/de stare (DCD, CTS, SYNC/aşteptare-sincronizare, BREAK/ABORT, eroare-de-ritm-la-transmisie/EOM). Pentru a obţine starea curentă a intrării $\overline{DCD}$, bitul DCD va trebui citit imediat după transmiterea comenzii 2, iniţializare-întreruperi-externe/de-stare.

SYNC/aşteptare-sincronizare indică, în modurile asincrone, starea intrării SYNC. În modurile sincrone, bitul se pune pe "0" în momentul sincronizării şi se poziţionează pe "1" prin programarea bitului aşteptare-sincronizare din WR3.

CTS indica starea conexiunii externe $\overline{CTS}$.

Eroare-de-ritm-la-transmisie/EOM pe "1" permite generarea întreruperii dacă a fost validată, la o transmisie într-un mod sincron, în momentul emisiei automate a caracterelor CRC, când *buffer*-ul de transmisie devine gol. Această întrerupere nu va fi generată dacă bitul D_6 este pe "0". $D_6=1$ și $D_2=0$ indică emisia unui caracter CRC, iar $D_6=1$ și $D_2=1$ indică emisia caracterelor SYNC.

BREAK/ABORT se pune pe "1" la detecția unei secvențe BREAK în modurile asincrone. După ce biții de stare vor fi inițializați prin trimiterea comenzii 2, bitul D_7 va putea trece pe "0" la terminarea secvenței BREAK. Dacă întreruperile externe/de stare sunt validate, schimbarea stării bitului D_7 va conduce la generarea întreruperii. În modul SDLC, D_7 se va pune pe "1" la detecția unei secvențe ABORT (cel puțin 7 biți "1").

Registrul de stare RR1 are formatul dat în continuare. După citirea acestui registru zona de adresă a registrului următor din WR0 se va pune pe 000.

D_7	D_6	D_5	D_4	D_3 D_2 D_1	D_0
Sfârșit de cadru (SDLC)	Eroare de CRC/ cadrare	Eroare depășire recepție	Eroare paritate	Codul de rest I la recepție	Toate caracterele transmise

Toate-caracterele-transmise este pus pe "1" în modurile asincrone, la transmisia completă a ultimului caracter din *buffer*-ul de transmisie. Modificarea acestui bit nu generează întrerupere. În modurile sincrone este întotdeauna "1".

Codul-de-rest-I-la-recepție indică lungimea câmpului de date, I, dintr-un cadru, în situațiile în care aceasta nu e un multiplu întreg de lungimea programată a caracterului:

D_3	D_2	D_1	Biți I în ultimul octet	Biți I în penultimul octet
1	0	0	0	3
0	1	0	0	4
1	1	0	0	5
0	0	1	0	6
1	0	1	0	7
0	1	1	0	8
1	1	1	1	8
0	0	0	2	8

Codul are semnificaţie numai în cazul transferurilor în care s-a poziţionat indicatorul sfârşit-de-cadru (SDLC). Codul specifică biţii de informaţie din ultimii doi octeţi preluaţi de unitatea centrală, biţii rămaşi din câmpul I fiind întotdeauna aliniaţi la dreapta în cadrul unuia din cei doi octeţi. Dăm mai jos valorile codului-de-rest-I-la- recepţie în cazul în care numărul de biţi/caracter a fost programat 8.

Atunci când limita ultimului caracter asamblat coincide cu limita câmpului I, începutul caracterelor CRC, valoarea codului va fi 011. Pentru număr de biţi/caracter diferiţi de 8 se pot construi tabele asemănătoare cu cel de mai sus.

Eroare-paritate va fi pus pe "1" pentru acele caractere a căror paritate calculată nu este egală cu paritatea recepţionată, evident cu condiţia de a valida paritatea, D_0 în WR4. Această eroare se memorează până la trimiterea unei comenzi 6, iniţializare-erori.

Eroare-depăşire-recepţie indică faptul că SIO a recepţionat mai mult de patru caractere fără ca UC să fi citit vreunul. Această eroare se ataşează numai caracterului înscris peste celelalte, dar, după citirea lui, ea se va memora până la trimiterea comenzii 6. Dacă s-a programat bitul starea-afectează-vectorul, încărcarea în stivă a caracterului care provoacă depăşire va conduce la generarea unei întreruperi, cu vectorul corespunzând condiţiei-de-recepţie-speciale.

Eroare-de-CRC/cadrare se pune pe "1" în modurile asincrone, dacă apare o eroare de cadrare, numai pentru caracterul greşit, fără a se memora în continuare. La detecţia unei erori de cadrare se adaugă, la durata unui caracter un timp corespunzând unei jumătăţi de bit, pentru ca eroarea să nu fie interpretată ca un nou bit de START. În modurile sincrone, bitul D_6 indică rezultatul verificării CRC.

Sfârşit-de-cadru (SDLC) indică în modul SDLC recepţia unui delimitator de sfârşit de cadru, eroare-de-CRC/cadrare şi codul-de-rest-I-la-recepţie devenind valide la poziţionarea pe "1" a acestui bit.

Registrul de stare RR2, citit numai de pe canalul B, conţine vectorul de întrerupere, la fel cu cel care a fost scris în WR2, dacă bitul starea-afectează-vectorul este "0", sau modificat, când acest bit este "1". Dacă SIO nu are întreruperi în aşteptare vectorul are biţii V_3, V_2 şi V_1 egali cu 011, ceilalţi fiind la fel cu cei programaţi în WR2. Formatul registrului RR2 este:

<table>
<tr><th>D_7</th><th>D_6</th><th>D_5</th><th>D_4</th><th>D_3</th><th>D_2</th><th>D_1</th><th>D_0</th></tr>
<tr><td>V_7</td><td>V_6</td><td>V_5</td><td>V_4</td><td>V_3</td><td>V_2</td><td>V_1</td><td>V_0</td></tr>
<tr><td colspan="4">La fel ca biţii din WR2</td><td colspan="3">Modificaţi dacă bitul starea-afectează-vectorul este "1"</td><td>La fel ca în WR2</td></tr>
</table>

2.4.4.3. Un exemplu de programare

În exemplul ce urmează se va presupune că SIO este selectat cu ajutorul bitului de adrese A_5 iar liniile $C/\overline{D}$ şi $B/\overline{A}$ sunt comandate cu biţii cei mai puţin semnificativi ai magistralei de adresă; adresele de I/E utilizate vor fi cele din tabelul:

Conexiuni externe			Adrese de I/E	Semnificaţia octetului transferat
$\overline{CE}(A_5)$	$C/\overline{D}(A_1)$	$B/\overline{A}(A_0)$		
0	0	0	20H	Date canal A
0	0	1	21H	Date canal B
0	1	0	22H	Comenzi/stări canal A
0	1	1	23H	Comenzi/stări canal B

Secvenţele de program date în continuare programează canalul B să lucreze asincron, iar canalul A în protocolul SDLC. Secvenţele urmează după o iniţializare hardware sau software.

```
ASYNC: LD A,18H           ; Iniţializare - canal B
       OUT (23H),A
       OUT (23H),A
       LD A,02H           ; Adresare registru WR2/B
       OUT (23H),A
       LD A,80H           ; Încărcare vector întrerupere
       OUT (23H),A
       LD A,04H           ; Adresare registru WR4/B
       OUT (23H),A
       LD A,47H           ; Rata datelor x 16 = Frecvenţa ceasului
       OUT (23H),A        ; 1 bit STOP, paritate pară
       LD A,05H           ; Adresare registru WR5/B
       OUT (23H),A
       LD A,2AH           ; 7 biţi/caracter la transmisie,
       OUT (23H),A        ; validare-transmisie, activare ieşire RTSB
       LD A,03H           ; Adresare registru WR3/B
       OUT (23H),A
       LD A,61H           ; 7 biţi/caracter la recepţie, auto-
       OUT (23H),A        ; validare, validare-recepţie
       LD A,01H           ; Adresare registru WR1/B
       OUT (23H),A
       LD A,17H           ; Întrerupere-la-fiecare-caracter-recep-
       OUT (23H)          ; ţionat/paritatea-afectează-vectorul,
                          ; validare-întreruperi-transmisie, vali-
                          ; dare-întreruperi-externe/de-stare
SDLC:  ...
       LD A,18H           ; Iniţializare - canal A
       OUT (22H),A
       OUT (22H),A
       LD A,04H           ; Adresare registru WR4/A
```

```
OUT (22H),A
LD A,20H        ; SDLC, Rata datelor x 1 = Frecvenţa
OUT (22H),A     ; ceasului, fără paritate
LD A,46H        ; Adresare registru WR6/A, iniţializare-
OUT (22H),A     ; verificare-CRC-la-recepţie
LD A,ADR        ; Încărcare adresă mesaj SDLC
OUT (22H),A
LD A,87H        ; Adresare registru WR7/A, iniţializare-
OUT (22H),A     ; generare-CRC-la-transmisie
LD A,7EH        ; Încărcare delimitator SDLC
OUT (22H),A
LD A,01H        ; Adresare registru WR1/A
OUT (22H),A
LD A,17H        ; Întrerupere-la-fiecare-caracter-recepţionat
OUT (22H),A     ; starea-afectează-vectorul, validare-
                ; întreruperi-transmisie,validare-întreruperi-
                ; externe/de-stare
LD A,15H        ; Adresare registru WR5/A, iniţializare-
OUT (22H),A     ; întreruperi-externe/de-stare
LD A,0E9H       ; DTR, 8 biţi/caracter la transmisie,
                ; validare-transmisie, validare-CRC-la-transmisie
OUT (22H),A
LD A,03H        ; Adresare WR3/A
OUT (22H),A
LD A,0EDH       ; 8 biţi/caracter recepţionat, auto-
OUT (22H),A     ; validare, validare-CRC-la-recepţie,
...             ; căutare-adresă, validare-recepţie
```

BIBLIOGRAFIE

1. * * *, Zilog, *1982/1983 Data Book.*
2. * * *, Mostek, *1979 Microcomputer Data Book.*
3. * * *, Mostek, *Z80 Microcomputer Software, Programming Guide.*
4. NICHOLS, E.A.;NICHOLS, J.C.; RONY, P.E., *Z-80, Livre 1, Programmation*, Publitronic Sarl, 1981.
5. NICHOLS, J.C.; NICHOLS, E.A.; RONY, P.E., *Z-80 Microprocessor, Programming & Interfacing, Book 2*, Howard W. Sams & Co. Inc., Indianapolis, 1979.
6. CARR,J.J., *Z-80 Users Manual*, Reston Publishing Company, Inc., Reston, Virginia, 1980.
7. MILLER, A.R., *8080/Z80 Assembly Language: Techniques for Improved Programming*, John Wiley & Sons, 1981.
8. WAGNER, W., *Sistemul de microprocesoare U880D*, RFT.
9. MUREŞAN, T.; STRUNGARU, C.; STOINESCU, R.; PETRIU E., *Microprocesorul 8080 în aplicaţii*, Editura Facla, Timişoara, 1981.
10. CĂPĂŢÎNĂ, O.D.; HAŞEGAN, M.C.; PUŞCĂ, M.V., *Proiectarea cu microprocesoare*, Editura Dacia, Cluj-Napoca, 1983.
11. PETRESCU, A. (coordonator), *Microcalculatoarele Felix M18, M18B, M118*, Editura Tehnică, Bucureşti, 1984.
12. PETRESCU, A. (coordonator), *Totul despre ... calculatorul personal aMIC*, Editura Tehnică, Bucureşti, 1985.
13. TOACŞE, G., *Introducere în microprocesoare*, Editura Ştiinţifică şi Enciclopedică, Bucureşti, 1985.
14. WEISSBERGER, A.J., *Data Communications Handbook*, Signetics Corporation, 1979.
15. COCULESCU, L.; POINARIU, C., *Teleprelucrarea datelor*, Editura Militară, Bucureşti, 1982.

16. DAVIES, D.W.; BARBER, D.L.A., *Rețele de interconectarea calculatoarelor*, Trad. din lb. engleză, după ediția a II-a, 1975, Editura Tehnică, București, 1976.
17. DAVIES, D.W.; BARBER, D.L.A.; PRICE, W.L.; SALMONIDES, C.M., *Computer Networks and Their Protocols*, John Wiley & Sons, 1979.
18. ROLLAND, B., *La liaison entre controleurs et imprimantes: les differentes interfaces*, Minis et micros, 1985, No. 240, p. 87-91.
19. * * *, *Intel Peripheral Design Handbook*.
20. * * *, *Intel 8080 Microcomputer Systems: User's Manual*, September 1975.
21. * * *, *Philips Data Handbook, Microprocessors, microcontrollers and peripherals*, Book IC14N, 1985.
22. NICHOLS, E.A., NICHOLS, J.C.; MUSSON, K.R., *Data Communications for Microcomputers. Practical Experiments for Z80 Microcomputers*, McGraw-Hill, 1983.
23. DOLL, D.R., *Data Communications: Facilities, Network and Systems Design*, John Wiley & Sons, 1978.
24. FOLTS, H.C., *McGraw-Hill Compilation of Data Communication Standards*, McGraw-Hill, 1981.
25. SEIDLER, J., *Principles of Computer Communication Network Design*, John Wiley & Sons, 1983.
26. LUPU, C.; ȚEPELEA, V.; PURICE, E., *Microprocesoare. Aplicații*, Editura Militară, București, 1982.
27. LUPU, C.; STĂNCESCU, Șt., *Microprocesoare. Circuite. Proiectare*, Editura Militară, București, 1986.
28. AUMIAUX, M., *Microprocesseurs 8 bits*, Masson,1985.
29. NICOUD, J.D., *Circuits numériques pour interfaces microprocesseur*, Masson, 1991.
30. AMGHAR, A., *Méthodes de développement d'un système à microprocesseurs*, Masson, 1989.
31. SEYER, M.D., *L'interface RS-232-C*, Masson, 1988.

Redactor: D. NICOLESCU
Coperta: VICTOR ILIE
Tehnoredactor: D. ANDREI
Apărut: 1995. Coli de tipar 11,75. B10395

Tiparul executat la tipografia editurii

Romfel SRL
Bucureşti
Str. Lizeanu nr. 35A, sect. 2

Patron unic şi Director general Cristian Rutăsini

Str. Lizeanu nr.35 A, sector 2

microprocesoare

Editura
Militară

ISBN 973-32-0409-9 LEI 6 000